ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

GENERACIÓN DE CALENDARIOS DE EXÁMENES MEDIANTE COLOREO DE GRAFOS

Lorena Gutiérrez Madroñal

Cádiz Diciembre 2007

ESCUELA SUPERIOR DE INGENIERÍA
INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

GENERACIÓN DE CALENDARIOS DE EXÁMENES MEDIANTE COLOREO DE GRAFOS

DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Directores del Proyecto: Francisco Palomo Lozano e Inmaculada Medina Bulo
Autor del Proyecto: Lorena Gutiérrez Madroñal

Cádiz Diciembre 2007
Fdo: Lorena Gutiérrez Madroñal.

Agradecimientos

Sería incorrecto decir que no tengo palabras para agradecer la dedicación y el apoyo de mis tutores Francisco Palomo Lozano e Inmaculada Medina Bulo. Ellos han seguido cada avance y cada tropiezo desde el principio hasta el final de mi Proyecto. Me han ayudado a dar el siguiente paso, ver aquello que no llegaba alcanzar y comprender lo que para en mis ojos había una venda. Claro que tengo palabras para ellos, pero por resumir emplearé una sola que las abarca todas: ¡GRACIAS!

Recuerdo las palabras de mi padre cuando tenía unos 15 años: "Por mucho que tú digas, yo te veo delante de un ordenador" y para variar, como todos los padres, tenía razón... a pesar de mis fracasos él siempre estuvo ahí; preocupándose por cada lágrima, admirando cada momento de esfuerzo... Se alegró más que nadie con mis éxitos (a veces hasta más que yo). No sólo seré feliz por mis esfuerzos al conseguir el terminar la carrera, aún lo seré más cuando vea en sus ojos el orgullo de verme conseguir mi meta. ¡Gracias papá!

Y como no mencionar a todos y cada uno de mis compañeros de Facultad; en algunos de ellos no sólo he encontrado el afán y dedicación a una vocación, los ánimos y nervios ante los exámenes y revisiones... el compañerismo y el trabajo en grupo... he encontrado amigos y profesionales. Gracias por dejarme aprender con vosotros, os deseo lo mejor para este gran paso que nos queda. Este proyecto no es solo mio, todos formais parte de él y de mi carrera.

Para mi gente, aquellos que siempre confiaron en mi y me apoyaron cuando lo necesitaba: ¡Cada paso no habría sido posible sin vosotros, gracias!, ¡Nunca dejaré de dároslas!

Índice general

1. Introducción **15**

1.1. Objetivos 15

1.2. Alcance 15

1.3. Definiciones, acrónimos y abreviaturas 17

1.4. Visión general 19

2. Desarrollo del calendario **21**

2.1. Fases de desarrollo del PFC 21

3. Descripción general del proyecto **25**

3.1. Perspectiva del producto 25

3.1.1. *Lenguaje C++* 25

3.1.2. *Librerías wxWidgets* 26

3.1.3. *Lenguaje Perl* 26

3.1.4. *Scripts y líneas de comando* 28

3.1.5. *Lenguaje LaTeX* 28

3.2. Funciones del producto 29

3.3. Características de los usuarios 29

3.4. Restricciones generales 30

3.5. Suposiciones y dependencias 30

3.6. Requisitos para futuras versiones 31

4. Desarrollo del Proyecto **33**

4.1. Estudio de Viabilidad del Sistema 34

4.1.1. Establecimiento del alcance del sistema 34

4.1.2. Estudio de la situación actual y predefinición de requisitos . . . 34

4.1.3. Estudio y valoración de alternativas de solución 35

4.2. Análisis del Sistema 37

4.2.1. Modelos de Casos de Uso 38

4.2.2. *Casos de uso asociados a los requisitos funcionales* 38

4.2.3. *Diagramas de los Casos de Uso* 39

4.2.4. *Especificación de Casos de Uso* 41

4.2.5. **Caso de uso PInicial** 42

4.2.6. **Caso de uso PFecha** 43

4.2.7. **Caso de uso PLista Titulaciones** 44

4.2.8. **Caso de uso PNueva** 45

4.2.9. **Caso de uso PAlgoritmos** 46

4.2.10. **Caso de uso PResultado** 47

4.2.11. **Caso de uso PAyuda** 48

4.2.12. Modelo conceptual de datos 49

4.2.13. Modelo de comportamiento del sistema 51

4.2.14. *Diagramas de secuencia de sistemas y Contratos de las operaciones del sistema* 52

4.2.15. Definición de Interfaces de Usuario 72

4.3. Diseño del sistema 80

4.3.1. Arquitectura del sistema software 80

4.3.2. Diseño de la capa de interfaz de usuario 81

4.3.2.1. *Diagramas de secuencia* 81

4.3.3. Diseño de la capa de control de la aplicación 88

4.3.3.1. *Diagramas de secuencia* 88

4.3.4. Diseño de la capa de dominio 93
4.3.4.1. *Diagramas de secuencia* 93
4.4. Pruebas y validación 97
4.4.1. Primera prueba 97
4.4.2. Segunda prueba 97
4.4.3. Tercera prueba 98
4.4.4. Cuarta prueba 98
4.4.5. Quinta prueba 98
4.4.6. Sexta prueba 99
4.4.7. Séptima prueba 99
4.5. Gráficos de las pruebas 100
4.6. Documentación de la ejecución de las pruebas 102
4.6.1. Datos Prueba1: 10 iteraciones 102
4.6.1.1. Algoritmo Devorador 102
4.6.1.2. Algoritmo BackTracking 102
4.6.1.3. Algoritmo Devorador Mejorado 102
4.6.2. Datos Prueba2: 50 iteraciones 103
4.6.2.1. Algoritmo Devorador 103
4.6.2.2. Algoritmo BackTrackin 103
4.6.2.3. Algoritmo Devorador Mejorado 103
4.6.3. Datos Prueba3: 100 iteraciones 104
4.6.3.1. Algoritmo Devorador 104
4.6.3.2. Algoritmo BackTracking 104
4.6.3.3. Algoritmo Devorador Mejorado 104
4.6.4. Datos Prueba4: 200 iteraciones 105
4.6.4.1. Algoritmo Devorador 105
4.6.4.2. Algoritmo BackTracking 105
4.6.4.3. Algoritmo Devorador Mejorado 105

4.6.5. Datos Prueba5: 500 iteraciones . . . 106
4.6.5.1. Algoritmo Devorador . . . 106
4.6.5.2. Algoritmo BackTracking . . . 106
4.6.5.3. Algoritmo Devorador Mejorado . . . 106
4.6.6. Datos Prueba6: 1000 iteraciones . . . 107
4.6.6.1. Algoritmo Devorador . . . 107
4.6.6.2. Algoritmo BackTracking . . . 107
4.6.6.3. Algoritmo Devorador Mejorado . . . 107
4.6.7. Datos Prueba7: 2000 iteraciones . . . 108
4.6.7.1. Algoritmo Devorador . . . 108
4.6.7.2. Algoritmo BackTracking . . . 108
4.6.7.3. Algoritmo Devorador Mejorado . . . 108
4.7. Pruebas adicionales . . . 109
4.7.1. 150 Iteraciones . . . 109
4.7.1.1. Algoritmo Devorador . . . 109
4.7.1.2. Algoritmo BackTracking . . . 109
4.7.1.3. Algoritmo Devorador Mejorado . . . 109
4.7.2. 300 Iteraciones . . . 110
4.7.2.1. Algoritmo Devorador . . . 110
4.7.2.2. Algoritmo BackTracking . . . 110
4.7.2.3. Algoritmo Devorador Mejorado . . . 110

5. Resumen **112**

6. Conclusiones **114**

7. Manual de Usuario **116**

7.1. **Calendarios paso a paso** . . . 116
7.1.1. **Antes de ejecuar la aplicación** . . . 116

7.1.2. **¡Calendarios!** . . . 118
7.1.3. **¡Fechas, fechas, fechas!** . . . 120
7.1.4. **Escoge la titulación** . . . 122
7.1.5. **¡Alguien nuevo!** . . . 126
7.1.6. **Los tres algoritmos** . . . 130
7.1.7. **Voilá!** . . . 135
7.1.8. **Help!, ¡Ayuda!, SOS** . . . 140

8. Manual de Instalación **142**
8.1. **Instalando wxWidgets** . . . 142
8.2. **Instalando recode** . . . 143
8.3. **Instalando LaTeX** . . . 143

9. Referencias **145**

10. Anexos **147**
10.1. Licencia GPL . . . 147
10.1.1. GNU GPL . . . 147
10.1.2. Validez legal . . . 147
10.2. GPL . . . 148
10.2.1. GNU GENERAL PUBLIC LICENSE (GPL) . . . 148
10.2.2. *Preámbulo* . . . 148
10.2.3. TÉRMINOS Y CONDICIONES PARA LA COPIA, DISTRIBUCIÓN Y MODIFICACIÓN . . . 149
10.3. Algoritmo Devorador . . . 156
10.4. Algoritmo BackTracking . . . 167
10.5. Algoritmo Devorador Mejorado . . . 174

Índice de figuras

1.1. Ejemplo Clique 18

2.1. Trayectoria del Proyecto 23

4.1. Diagrama general de los casos de uso 39
4.2. Diagrama del caso de uso PInicial, la pantalla Inicial 39
4.3. Diagrama del caso de uso PFechas, la pantalla Inserción de datos . . . 40
4.4. Diagrama del caso de uso PTitulaciones, la pantalla Lista de titulaciones 40
4.5. Diagrama del caso de uso PAlgoritmos, la pantalla Algoritmos 40
4.6. Diagrama del caso de uso PResultado, la pantalla Resultado 40
4.7. Modelo Conceptual de Datos 50
4.8. Diagrama de Comportamiento PInicial 52
4.9. Diagrama de Comportamiento Llamada Ayuda 53
4.10. Diagrama de Comportamiento PFecha 54
4.11. Diagrama de Comportamiento Llamada Ayuda 55
4.12. Diagrama de Comportamiento PLista, operación Siguiente 56
4.13. Diagrama de Comportamiento PLista, operación Buscar 57
4.14. Diagrama de Comportamiento PLista, operación Nueva 58
4.15. Diagrama de Comportamiento Llamada Ayuda 59
4.16. Diagrama de Comportamiento PNueva, operación Cerrar 60
4.17. Diagrama de Comportamiento PNueva, operación Guardar 61
4.18. Diagrama de Comportamiento Llamada Ayuda 62

4.19. Diagrama de Comportamiento PAlgoritmos, operación Aceptar 63
4.20. Diagrama de Comportamiento PNueva, operación Elección 64
4.21. Diagrama de Comportamiento Llamada Ayuda 65
4.22. Diagrama de Comportamiento PResultado, operación Atrás 66
4.23. Diagrama de Comportamiento PResultado, operación Imprimir 67
4.24. Diagrama de Comportamiento PResultado, operación Imprimir 68
4.25. Diagrama de Comportamiento Llamada Ayuda 69
4.26. Diagrama de Comportamiento PAyuda, operación Cerrar 70
4.27. Diagrama de Comportamiento PAyuda, operación IrA 71
4.28. Pantalla de Inicio 72
4.29. Pantalla de Ayuda 73
4.30. Pantalla de Insercción de datos 74
4.31. Pantalla de Lista de Titulaciones 75
4.32. Pantalla Nueva Titulación 76
4.33. Pantalla de Algoritmos 77
4.34. Pantalla Resultado 78
4.35. Diagrama de Interacción PInicial 81
4.36. Diagrama de Interacción PFechas 82
4.37. Diagrama de Interacción PLista Titulaciones 83
4.38. Diagrama de Interacción PNueva 84
4.39. Diagrama de Interacción PAlgoritmos 85
4.40. Diagrama de Interacción PResultado 86
4.41. Diagrama de Interacción PAyuda 87
4.42. Diagrama de Interacción, capa de control en Pantalla Fechas 88
4.43. Diagrama de Interacción, capa de control en Pantalla Lista Titulaciones 89
4.44. Diagrama de Interacción, capa de control en Pantalla Nueva 90
4.45. Diagrama de Interacción, capa de control en Pantalla Algoritmos 91
4.46. Diagrama de Interacción, capa de control en Pantalla Resultado 92

4.47. Diagrama de Interacción, capa de dominio, Algoritmo 1 Devorador . . . 93

4.48. Diagrama de Interacción, capa de dominio, Algoritmo 2 BackTracking . 94

4.49. Diagrama de Interacción, capa de dominio, Algoritmo 3 Devorador Mejorado 95

4.50. Estudio del tiempo en el algoritmo Devorador 100

4.51. Estudio del tiempo en el algoritmo BackTracking 101

4.52. Estudio del tiempo en el algoritmo Devorador Mejorado 101

7.1. Pantalla de Inicio 118

7.2. Pantalla de Información 119

7.3. Pantalla de Introducción de datos 120

7.4. Pantalla Fechas erróneas 121

7.5. Pantalla Lista de Titulaciones 122

7.6. Mensaje, Titulación no encontrada 123

7.7. Mensaje, Titulación encontrada 124

7.8. Mensaje, Titulación no seleccionada 125

7.9. Mensaje, ¡Sin espacios! 126

7.10. Pantalla nueva Titulación 127

7.11. Introducimos nueva Titulación 128

7.12. Pantalla Lista de Titulaciones actualizada 129

7.13. Procesamiento de datos 130

7.14. Pantalla inicial de Algoritmos 131

7.15. Algoritmo 1, Devorador 132

7.16. Algoritmo 2, BackTracking 133

7.17. Algoritmo 3, Devorador Mejorado 134

7.18. Cálculo terminado del Algoritmo 1, Devorador 135

7.19. Cálculo terminado del Algoritmo 2, BackTracking 135

7.20. Cálculo terminado del Algoritmo 3, Devorador Mejorado 136

7.21. Calendario, solución 137

7.22. Mensaje de impresión 138
7.23. Calendario, solución PDF 139
7.24. Calendario, solución 140

Capítulo 1

Introducción

1.1. Objetivos

El objetivo principal de nuestro Proyecto es el de generar un calendario de exámenes para las diferentes titulaciones de modo que obtengamos un calendario en el que no coincidan los exámenes de las asignaturas que tienen alumnos matriculados en común. De esta forma nos podemos evitar "la segunda vuelta" de los mismos, consiguiendo así que sólo se realice un único exámen, que no existan disputas por la dificultad de las diferentes vueltas y que los profesores sólo elaboren un examen.

1.2. Alcance

Todos hemos sufrido "la segunda vuelta" de un examen debido a la coincidencia de asignaturas. Para los profesores supone el elaborar un nuevo examen que tenga la misma dificultad que el que anteriormente habían pensado y para los alumnos supone el alargar sus días de estudio y una modificación en sus calendarios de exámenes. Resumiendo, todos perdemos con la segunda vuelta de los exámenes y con el producto "Calendarios" podemos evitarlo.

Para conseguir nuestro objetivo necesitamos saber los alumnos matriculados. Lo que supone un problema ya que estos no se matriculan hasta el mes de Octubre y el calendario de los exámenes tiene que publicarse en el mes de Julio, con lo cual se nos hace imposible conocer el número de matriculados en las asignaturas. Este será nuestro objetivo a resolver para el PFC de Ingeniero en Informática dentro de unos años, que resolveremos haciendo un estudio estadístico sobre las matriculaciones en años anteriores (aproximadamente dos años) en las diferentes asignaturas.

Nuestro calendario sólo nos muestra una optimización de la distribución de las asignaturas en el intervalo de los días que le indicamos, no nos asigna un aula; suponemos que disponemos de aulas suficientes a la hora de asignar los días de examen a las

asignaturas.

Los beneficios que obtenemos es el evitarnos la segunda vuelta y todo lo que conlleva esto: elaboración de un segundo tipo de examen de igual dificultad, alargar y modificar el calendario de exámenes de los alumnos, pensar en otro día posible para dicha segunda vuelta... De este modo facilitamos el trabajo de muchas personas cuando tienen que crear el calendario de los exámenes de tal forma que sea óptimo para todos.

"Calendarios" nos ofrece tres posibles resultados, el primero nos da una solución pero en un tiempo muy corto de tiempo, el segundo nos da la solución óptima del calendario pero en un tiempo mucho mayor y el tercero es una optimización del primero, basándonos en el resultado del primero, obtenemos una mejora de la solución anterior.

Nuestro producto "Calendiarios" es aplicable para generar los exámenes de cualquier titulación y nos ofrece la posibilidad de añadir una nueva, si esta no existiera en la aplicación. Se puede ejecutar en cualquier plataforma UNIX sin ningún problema, con lo que hace de nuestro programa un producto transportable.

Como hemos comentado, nuestra meta es mejorar el producto "Calendarios" en el PFC de Ingeniero en Informática, de tal forma que podamos contemplar sin ningún problema los alumnos matriculados haciendo un estudio estadístico de las matriculaciones en años anteriores.

1.3. Definiciones, acrónimos y abreviaturas

Coloración por vértices: Dado un conjunto de colores C, una coloración por vértices de un grafo G = (V, E) es una función

$$f : V \rightarrow C$$

tal que si

$$uv \in E$$

entonces

$$f(u) \neq f(v)$$

En otras palabras, una coloración por vértices de un grafo es una asignación de colores a los vértices de manera tal que a vértices adyacentes les correspondan colores diferentes. En realidad los elementos del conjunto C pueden ser objetos de cualquier naturaleza y en última instancia lo único que interesa es su número, pero se usa la metáfora de los colores porque es intuitiva y fácil de comprender.

Grado de un grafo: Sea el grafo G(V, A), donde

$$V = \{a, b, c, d, e\}$$

y

$$A = \{(a, c), (d, e), (a, e), (b, e), (c, a), (c, c), (d, b)\}$$

El grado (positivo o negativo) de un vértice v (y se indica como (v)), como la cantidad de aristas que llegan o salen de él; para el caso de grafos no orientados, el grado de un vértice es símplemente la cantidad de aristas que tocan este véretice. Por ejemplo, el grado positivo (salidas) de d es 3, mientras que el grado negativo (llegadas) de b es 1.

El grado de un grafo es el de su vértice de máximo grado.

Número cromático de un grafo: Es el menor número natural k entra todos los valores posibles que permiten k-colorear un grafo. Se denomina a esta valor como χ(G).

Grafo k-colorable: Un grafo G es k-colorable si admite una coloración con k colores. Véase "Número cromático de un grafo".

Obviamente todo grafo de orden n es n-colorable. K_n no se puede colorear con menos de n colores, pues como todos sus vértices son adyacentes cada uno debe pintarse de un color difernte. Por lo tanto $\chi(k_n) = n$.

El número cromático de un grafo sin aristas es 1. Un camino de longitud $n > 2$ tiene número cromático 2, ya que sus vértices pueden pintarse con dos colores en forma alternada, comenzando por un extremo.

Clique: Una clique en un grafo es un conjunto de vértices dos a dos adyacentes. En el grafo del dibujo, los vértices 1, 2 y 5 forman una clique porque cada uno tiene

un arco que le une a los otros. En cambio, los vértices 2, 3 y 4 no, dado que 2 y 4 no son adyacentes.

El problema de la clique un problema NP-completo según la Teoría de la complejidad computacional. Es un problema de optimización para determinar cuándo un grafo contiene una clique de al menos un tamaño k. Una vez que tenemos k o más vertices que forman una clique, es trivial verificar que lo son, por eso es un problema NP. El correspondiente problema de optimización, consiste en encontrar una clique de tamaño máximo en un grafo (un subgrafo completo de tamaño máximo). Este problema se puede enunciar como un problema de decisión si la pregunta que se hace es saber si existe una clique de tamaño k en el grafo.

$Clique$ ={$(G, k)|G$}; tamaño $\geq$ k

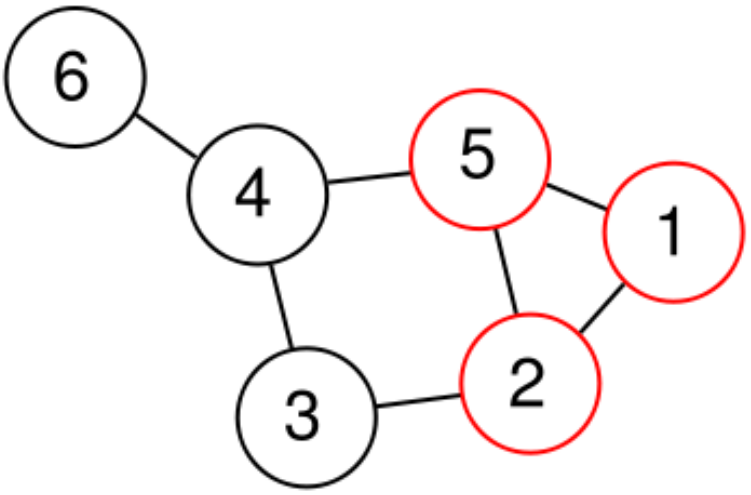

Figura 1.1: Ejemplo Clique

1.4. Visión general

La MPFC continúa con el proceso de desarrollo del PFC, seguimiento y esfuerzo de las diferentes partes, se realiza una descripción general del proyecto donde ampliaremos la información sobre el mismo: perspectivas, funciones del producto, usuarios a los que va dirigido, restricciones, suposiciones y dependencias del mismo, requisitos para futuras versiones... Finalmente se expondrá el desarrollo del proyecto en el que, al estar hablando de un Proyecto de Desarrollo de Aplicaciones Software, se incluirán los siguientes apartados: metodología de desarrollo, especificación de los requisitos del sistema, análisis del sistema, diseño del sistema, la codificación, las pruebas y la validación, el resumen, las conclusiones, el manual de usuario, el manual de instalación, las referencias y los anexos necesarios.

Capítulo 2

Desarrollo del calendario

2.1. Fases de desarrollo del PFC

A continuación se especifican y enumeran las distintas tareas que se han llevado a cabo para la elaboración del PFC. Estas tareas las representaremos mediante bloques y, como sabemos, se pueden solapar en el tiempo y no se tiene que terminar completamente la elaboración de una para el comienzo de la siguiente ya que no son actividades críticas.

Gracias al "Diagrama de Gantt" se puede ver graficamente y de una manera clara el solapamiento del que hablamos.

La elaboración total del proyecto ha tenido un coste en tiempo aproximado de 14 meses (desde Octubre 2006 hasta Diciembre 2007) aunque se verá posteriormente que la dedicación al mismo no ha sido plena, debido a que en el primer cuatrimestre y un poco en el segundo se compaginaba con la finalización de las últimas asignaturas de la diplomatura.

1. En este primer bloque se analiza y estudia el abanico de posibilidades (lenguajes de programación, entornos, herramientas...) que teníamos a disposición. Es decir, qué lenguaje de programación usar (Java, C++, etc), herramientas disponibles, wxFormBuilder, emacs, etc. A la par que decidimos los algoritmos a implementar que generarán nuestro calendario (BackTrackin, Devorador, Devorador Mejorado, etc).

   A modo de reseña mostramos las opciones escogidas:

   a) Lenguaje c++

   b) Librerías wxWidgets

   c) Lenguaje Perl

   d) Scripts y operaciones del sistema

   e) Lenguaje LaTeX Herramientas:

f) emacs Algoritmos:

g) Devorador

h) BackTracking

i) Devorador Mejorado

A posteriori se explicarán de una forma más detallada cada uno de los lenguajes y algoritmos empleados.

2. Una vez escogidas las herramientas y entornos concretos, en este segundo bloque se aborda el aprendizaje y análisis de las principales características de los diferentes algoritmos como de los nuevos lenguajes que no tenemos conocimiento (véase el caso de wxWidgets) y el perfeccionamiento de los lenguajes ya conocidos.

   Este bloque se prolonga casi a lo largo de todo el desarrollo del proyecto y podríamos decir que aún se continúa. El aprendizaje ha sido progresivo y de menos a más hasta establecer una técnica de programación en la que, salvo pequeñas puntualizaciones, errores, etc. ha sido constante.

3. En el tercer bloque, al no tener referencia sobre ningún programa que realice nuestro objetivo, tenemos que analizar de forma detallada, basándonos en nuestra propia experiencia, los requisitos (tanto funcionales como no funcionales) del sistema. Se ha seguido un modelo evolutivo, más concretamente en espiral, donde se obtuvieron unos requisitos iniciales que posteriormente se fueron incrementando hasta el producto final. Este bloque se alarga en casi todo el desarrollo del proyecto. Menos en la parte final donde ya se tiene definido claramente y el bloque finaliza.

4. En este cuarto bloque abordamos la implementación de los algoritmos y la adaptación de estos a la aplicación desarrollada, los algoritmos: algoritmo devorador, algoritmo backtracking y algoritmo devorador mejorado.

5. El quinto bloque es para la elaboración de la interfaz del programa, la cual a medida que estudiábamos los requisitos de la aplicación, iba alargándose hasta llegar a la solución final.

6. El bloque sexto consiste en la aplicación de una batería de pruebas que certifiquen la aplicación final cumple con los requisitos iniciales. En esta fase se realizan las modificaciones pertinentes acorde al resultado de las pruebas realizadas.

7. Respecto a la documentación de esta memoria (séptimo último bloque), mencionar que la realización de la misma ha sido escalonada. Es decir, después de cada análisis, elección o modificación, ésta ha sido modificada con respecto el tema que se trataba en esa etapa del proyecto.

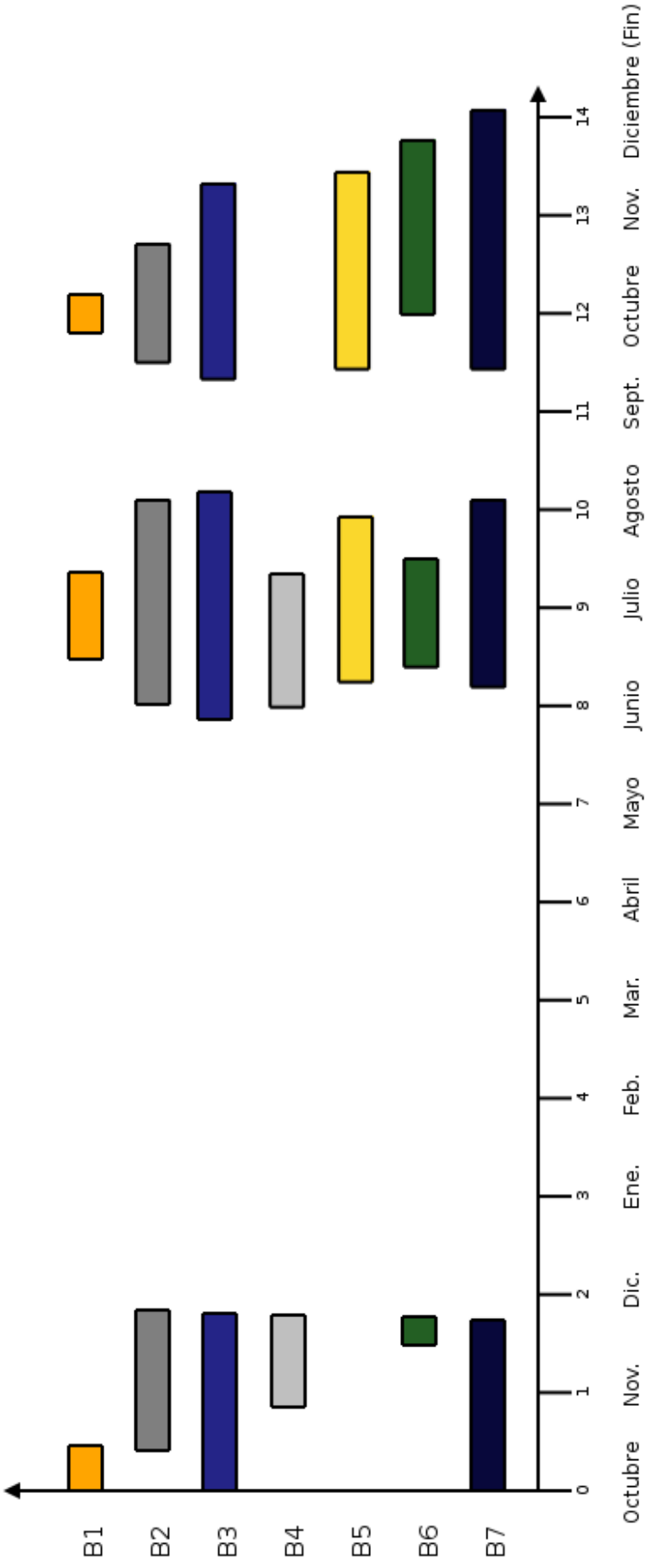

Figura 2.1: Trayectoria del Proyecto

Capítulo 3

Descripción general del proyecto

3.1. Perspectiva del producto

Nuestro producto "Calendarios" es un proyecto independiente que, como ya hemos comentado, tiene intención de evolucionar mejorando en su análisis y en sus soluciones.

3.1.1. *Lenguaje C++*

"Calendarios" está implementado bajo el lenguaje de programación C++, con lo que contamos con las siguientes ventajas:

- Es un lenguaje que permite programar de forma estructurada, modular y orientado a objetos, por lo que permite la asimilación de todas estas técnicas.
- Es un lenguaje que permite el manejo de todo tipo de estructuras de datos (arrays, pilas, colas, textos, objetos...) por lo que es capaz de resolver todo tipo de problemas.
- Es un lenguaje compilado, lo que le hace muy rápido (quizá el más rápido descontando el ensamblador).
- Permite crear todo tipo de aplicaciones (aunque no está pernsado para crear aplicaciones para ejecutar desde Internet).
- Gracias a su capacidad de uso de objetos y clases, facilita la reutilización del código.
- Permite incorporar las librerías más utilizadas para hacer aplicaciones.
- Es el lenguaje que sirve de base para los lenguajes modernos como Java, c♯, Perl...
- Lenguaje muy eficiente puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas.
- A pesar de su bajo nivel es el lenguaje más portado en existencia, habiendo compiladores para casi todos los sistemas conocidos.
- Proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes.

3.1.2. *Librerías wxWidgets*

La interfaz del producto "Calendarios" se ha realizado bajo wxWidgets, un framework antes conocido como wxWindows, especializado en el desarrollo de aplicaciones multiplataforma en lenguaje C++. Fue en definitiva el que escogí para realizar mi aplicación, por las siguientes razones:

- Soporta varios sistemas operativos, pero con una pequeña particularidad: de acuerdo a la plataforma en que se ejecute la aplicación, esta tomara el look and feel del SO donde se esté ejecutando. Por ejemplo, si se ejecuta en MS-Windows, la aplicación tendrá la apariencia de una aplicación hecha con el API de Windows, pero si se ejecuta en GNU/Linux, tendrá la apariencia de una aplicación GTK2. Es decir que la parte de GUI es solo una capa para el API nativa de cada SO. Además de lo anterior, wxWidgets cuenta con una versión embebida para dispositivos como PDAs y teléfonos celulares.
- Es cien por ciento libre en todas las plataformas que soporta, con una licencia LGPL modificada.
- Tiene un API orientada a objetos que es, en extremo, fácil de aprender y utilizar.
- Posee funcionalidades para muchos aspectos aparte de la construcción de interfaces gráficas. Como gráficos 2D, 3D con OpenGL, Bases de Datos (ODBC), Redes, Impresión, Hilos, entre muchas otras.

Pese a todas estas ventajas, wxWidgets también tiene algunas desventajas: Su diseño orientado a objetos no es el mejor que uno pueda ver. A veces abusa de la utilización de macros para realizar ciertas operaciones (como las tablas de eventos), aunque esto hace que codificar sea más fácil, también complica la labor de depuración, especialmente para los mismos desarrolladores de wxWidgets. Por otro lado, debido a que el framework se comenzó a desarrollar hace bastante tiempo, no cuenta con soporte para algunas características relativamente modernas del lenguaje C++ como por ejemplo el manejo de excepciones, y la STL.

Hemos empleado el lenguaje Perl para la lectura interna de las páginas webs de las asignaturas de la titulación escogida. De esta forma hemos seleccionado los datos que necesitábamos, véase el nombre y código de la asignatura, y la información de los alumnos matriculados en ella (DNI de los alumnos) y así manejarlos según nuestras necesidades. Los porqués de esta selección de lenguajes los veremos a continuación, después de realizar una descripción del lenguaje.

3.1.3. *Lenguaje Perl*

Perl, Lenguaje Práctico para la Extracción e Informe (Practical Extraction and Report Languaje) es un lenguaje de programación diseñado por Larry Wall creado en 1987. Perl toma características del C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, muchos otros lenguajes de programación.

Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.

Perl es software libre y está licenciado bajo la Licencia Artística y la GNU General Public License. Existen distribuciones disponibles para la mayoría de sistemas operativos. Está especialmente extendido en Unix y en sistemas similares, pero ha sido portado a las plataformas más modernas (y otras más obsoletas).

Características a tener en cuenta del lenguaje:

- La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas.
- Perl también toma características de la programación shell. Todas las variables son marcadas con un signo precedente (sigil). Los sigil identifican inequívocamente los nombres de las variables, permitiendo a Perl tener una rica sintaxis. Notablemente, los sigil permiten interpolar variables directamente dentro de las cadenas de caracteres (strings). Como en los shell, Perl tiene muchas funciones integradas para tareas comunes y para acceder a los recursos del sistema.
- Perl toma las listas del Lisp, hash (memoria asociativa) del AWK y expresiones regulares del sed. Todo esto simplifica y facilita todas las formas del análisis sintáctico, manejo de texto y tareas de gestión de datos.
- En Perl 5, se añadieron características para soportar estructuras de datos complejas, funciones de primer orden (p. e. clausuras como valores) y un modelo de programación orientada a objetos. Éstos incluyen referencias, paquetes y una ejecución de métodos basada en clases y la introducción de variables de ámbito léxico, que hizo más fácil escribir código robusto (junto con el pragma strict). Una característica principal introducida en Perl 5 fue la habilidad de empaquetar código reutilizable como módulos. Larry Wall indicó más adelante que "la intención del sistema de módulos de Perl 5 era apoyar el crecimiento de la cultura Perl en vez del núcleo de Perl".
- Todas las versiones de Perl hacen el tipado automático de datos y la gestión de memoria. El intérprete conoce el tipo y requerimientos de almacenamiento de cada objeto en el programa; reserva y libera espacio para ellos según sea necesario. Las conversiones legales de tipo se hacen de forma automática en tiempo de ejecución; las conversiones ilegales son consideradas errores fatales.

Perl se usa a menudo como un "lenguaje pegamento", ligando sistemas e interfaces que no fueron diseñados específicamente para interoperar; y para el .[es]carbado de datos", convirtiendo o procesando grandes cantidades de datos para tareas como por ejemplo crear informes. De hecho, estas fortalezas están íntimamente unidas. Su combinación hace a Perl una popular herramienta de propósito general para los administradores de sistemas, especialmente en programas pequeños que pueden ser escritos y ejecutados en una sola línea de comandos.

3.1.4. *Scripts y líneas de comando*

También hacemos empleo del lanzamiento de scripts para realizar en paralelo algunas operaciones que nos harán falta a lo largo del programa importando y exportando información del mismo para realizar estas operaciones y para obtener datos de las operaciones paralelas. Veamos con detenimiento qué es un script y sus características:

Línea de comandos, Intérprete de mandatos, Intérprete de línea de mandatos, Intérprete de comandos, Terminal, Consola, Shell ó su acronimo en inglés CLI por Command line interface, es un programa informático que actúa como Interfaz de usuario para comunicar al usuario con el sistema operativo mediante una ventana que espera comandos textuales ingresados por el usuario en el teclado, los interpreta y los entrega al sistema operativo para su ejecución. La respuesta del sistema operativo puede ser mostrada al usuario en la misma ventana (dependiendo de los comandos). A continuación, la shell queda esperando más instrucciones.

Dada la importancia de esta herramienta, existe ya desde los comienzos de la computación. Existen para diversos sistemas operativos, diversos hardware, con diferente funcionalidad. Suelen incorporar características tales como control de procesos, redirección de entrada/salida, listado y lectura de ficheros, protección, comunicaciones y un lenguaje de órdenes para escribir programas por lotes o (scripts o guiónes).

El guión o archivo de procesamiento por lotes (en inglés script) es un programa usualmente simple, que generalmente se almacena en un archivo de texto plano. Los guiones son casi siempre interpretados, pero no todo programa interpretado es considerado un guión. El uso habitual de los guiones es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. Por este uso es frecuente que los shells sean a la vez intérpretes de este tipo de programas.

3.1.5. *Lenguaje LaTeX*

Para la realización de esta memoria y para la estructuración y diseño del resultado de nuestro programa "Calendarios" empleamos el lenguaje LaTeX para generar las tablas resultantes del producto, quedando con un entorno agradable y elegante.

Es un lenguaje de marcado para documentos, y un sistema de preparación de documentos, formado por un gran conjunto de macros de TeX, escritas inicialmente por Leslie Lamport (LamportTeX) en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica creado por Donald Knuth. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX es software libre bajo licencia LPPL.

El LaTeX es un procesador de textos que está formado mayoritariamente por órdenes (macros) construidas a partir de comandos de TeX lenguaje «de bajo nivel», en el sentido de que sus acciones últimas son muy elementales pero con la ventaja añadida, en palabras de Lamport, de «poder aumentar las capacidades de LaTeX utilizando comandos propios del TeX descritos en The TeXbook» . Esto es lo que convierte a LaTeX en una herramienta práctica y útil pues, a su facilidad de uso, se une toda la potencia de TeX. Estas características hicieron que LaTeX se extendiese rápidamente entre un amplio sector científico y técnico, hasta el punto de convertirse en uso obligado en comunicaciones y congresos, y requerido por determinadas revistas a la hora de entregar artículos académicos.

3.2. Funciones del producto

Como ya hemos comentado las principales funciones de nuestro PFC "Calendarios", son las de facilitar el trabajo y evitar las más que probables coincidencias de exámenes con las consecuentes segundas vueltas. Evitando así la elaboración de un segundo examen de igual dificultad que el anteriormente pensado, la modificación del calendario de exámenes de los alumnos...

"Calendarios" nos dará la opción de insertar una nueva titulación en el caso de que ésta no esté en la aplicación y nos permitirá indicar el día de comienzo y de finalización de nuestro calendario de exámenes, en el que recalcamos que el rango que consideramos en nuestro programa es desde el día de inicio que nos indican, hasta el día anterior al final que nos señalan como fin de calendario.

Se nos ofrecerán tres opciones para obtener el calendario: Una rápida, que nos da una solución en un periodo corto de tiempo, una óptima, que en un período más largo de tiempo nos ofrece la solución óptima al problema y una tercera que es una mejora de la primera solución; optimiza la solución del primer algoritmo a cambio de incrementar el tiempo de resolución.

3.3. Características de los usuarios

El tipo de usuarios a los que está dirigida la aplicación "Calendarios" serán los encargados de la creación de los calendarios de exámenes de las diferentes titulaciones. Estos deberán tener un conocimiento básico de ofimática que les permita manejarse por el S.O. establecido en su terminal de trabajo sin dificultades para ejecutar esta aplicación.

3.4. Restricciones generales

Sobre las restricciones de la aplicación "Calendarios" comentamos que sólo hay que seguir los pasos que se van explicando en la "Ayuda" de la aplicación que puede seguirse a la vez que estamos ejecutando la misma o en el manual de usuario.

Al estar implementada con wxWidgets no hay problemas con el S.O. ya que como hemos comentado soporta varios SS.OO. aunque en nuestro caso al lanzar scrips, no podremos utilizarlo en un entorno Windows.

En el manual de usuario le especificamos a nuestro usuario los pasos que tiene que ir dando para poder ejecutar de forma correcta la aplicación.

Existe un problema en nuestro producto y es que al no poder tener acceso a la Base de Datos principal donde están los datos de los alumnos, hemos simulado nuestra base de datos accediendo a la página web del "red-campus" de la UCA, en la que (siendo un usuario de la misma, de tal forma que podamos acceder a la lista de alumnos de una determinada asignatura) nos bajaremos las páginas correspondientes de las asignaturas de las que queramos realizar el calendario (obviamente serán de una misma titulación, si no, no tendría sentido). Estas páginas serán descargadas (le indicaremos el modo al usuario) y almacenadas en la carpeta "Asignaturas" de tal forma que luego el programa pueda hacer, basándose en los datos de las éstas, los calendarios de dichas asignaturas de una determinada titulación.

3.5. Suposiciones y dependencias

Independientemente del sistema operativo en el que utilicemos nuestro producto tendremos que instalar las librerías wxWidgets para dicho sistema (a no ser que ya estén incluidas). También necesitaremos para ejecutar nuestros scripts paralelos que tengan instalados Perl (aunque en los sistemas UNIX vienen por defecto), el paquete recode, para el manejo de datos de los ficheros, que tengan así una codificación UTF-8 y los paquetes correspondientes para el lenguaje LaTex para poder hacer la impresión de los resultados con este formato y posteriormente a PDF con los paquetes que también tienen que instalarse la aplicación pdflatex.

Dependemos, como ya hemos comentado antes, de la página web del "red-campus" de la UCA, ya que nos es imposible tener acceso a la Base de Datos de los alumnos. Tendríamos que esperar a que estas estén completas para poder hacer un calendario que se adecue más a los datos reales de las asignaturas. Teniendo conocimiento así de todos los datos necesarios de cada asignatura (alumnos matriculados en éstas). Podríamos hacer un calendario con los datos actuales para el curso siguiente, al igual que podríamos haber hecho un calendario de exámenes basándonos en los datos del año pasado (aunque con sólo esa muestra de datos no tendríamos una buena aproximación para nuestro calendario resultante).

3.6. Requisitos para futuras versiones

Para nuestra futura versión, necesitaremos el análisis estadístico de las matriculaciones anteriores (dos años mínimo). Implementaremos nuevos algoritmos, que dependiendo de las necesidades de los mismos necesitaremos unos u otros requisitos.

Intentaremmos, aunque es algo poco probable, tener acceso a la Base de datos de los alumnos, por lo menos la de años anteriores para hacer esa estadística ya comentada. Y así no depender de la página web de la UCA "red-campus".

Capítulo 4

Desarrollo del Proyecto

La elaboración de un producto software de calidad requiere la aplicación, durante todo su desarrollo, de una metodología. Aunque no existe una definición estricta del concepto *metodología de desarrollo*, ésta puede considerarse como un conjunto de pasos y procedimientos que deben seguirse para desarrollar un producto software.

Para el desarrollo de este proyecto se han seguido las recomendaciones de **Métrica Versión 3**: Metodología de Planificación, Desarrollo y mantenimiento de sistemas de información, propuesta por el Ministerio de Administraciones Públicas.

METRICA Versión 3 ha sido concebida para abarcar el desarrollo completo de Sistemas de Información sea cual sea su complejidad y magnitud, por lo tanto su estructura responde a desarrollos máximos y debe adaptarse y dimensionarse en cada momento de acuerdo a las características particulares de cada proyecto. En este capítulo se incluyen los modelos considerados más relevantes del proceso de Desarrollo de Sistemas de Información, que incluye los siguientes subprocesos:

a) Estudio de Viabilidad del Sistema (EVI)

b) Análisis del Sistema de Información (ASI)

c) Diseño del Sistema de Información (DSI)

d) Construcción del Sistema de Información (CSI)

e) Implantación y Aceptación del Sistema (IAS)

4.1. Estudio de Viabilidad del Sistema

4.1.1. Establecimiento del alcance del sistema

Debemos estudiar el alcance de la necesidad planteada por el cliente o usuario; en nuestro caso, el Proyecto fue planteado por uno de mis tutores en el transcurso de una de sus clases. Tras presentar mi interés sobre el tema, ambos tutores me situaron ante el problema general.

El objetivo final es la simulación por ordenador de un programa capáz de generar calendarios de exámenes óptimos, entendiendose óptimos por aquellos que reflejen las mejores fechas sin coincidencias, evitandose así la segunda convocatoria de los mismos. En el caso que hubiese, se buscarían aquellas que afectasen menos a los alumnos y profesores, es decir, aquellas que tienen menor número coincidente de matriculados. La generación de estos calendarios se hará mediante tres algoritmos diferentes; un algoritmo devorador que nos dará una solución rápida sin tener que ser la más óptima, un algoritmo de backtraking que nos dará la solución óptima aunque tardando más que el algoritmo anterior y un algoritmo de optimización del primero, aumentamos el tiempo de respuesta, pero optimizamos la solución anterior.

4.1.2. Estudio de la situación actual y predefinición de requisitos

Actualmente no existe ningún programa o aplicación que permita generar calendarios de exámenes para las universidades, institutos, colegios u otras organizaciones. Navegando por internet no encontramos nada referente a este tema, con lo que consideramos que todos estos organimos realizan esta tarea con el mismo procedimiento: “A mano”. Debido a esto, como ya hemos podido comprobar, surgen las coincidencias de exámenes con las consecuentes segundas vueltas y modificación del calendario de exámenes.

Para llevar a cabo el desarrollo de nuestro producto, nos es necesario conocer los alumnos matriculados en las asignaturas. Esto a día de hoy nos es imposible saber ya que el calendario de los exámenes se publica con varios meses de antelación respecto la matriculación de los alumnos. Por eso en la siguiente versión de este Proyecto ampliaremos no sólo con más algoritmos de coloreo de grafos, si no también con un estudio estadístico de años anteriores para que nuestros calendarios resultantes sean lo más exactos posibles.

4.1.3. Estudio y valoración de alternativas de solución

Al no tener referencia de otras aplicaciones, no tenemos otra alternativa que el desarrollo de este producto. Se nos plantea el lenguaje con el que implementarlo y pensamos en dos alternativas:

a) Java, que englobaría la implementación del programa e interfaz del mismo.

b) C++ y wxWidgets, el primero para la implementación y el segundo para la interfaz.

A pesar de que Java se presentaba como un lenguaje de programación más compacto para la implementación y la interfaz, escogimos C++ con wxWidgets, porque son lenguajes libres. Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems. Sun Microsystems proporciona una implementación GNU General Public License de un compilador Java y una máquina virtual Java, conforme a las especificaciones del Java Community Process, aunque la biblioteca de clases que se requiere para ejecutar los programas Java no es software libre.

Las wxWidgets son unas bibliotecas multiplataforma y libres, para el desarrollo de interfaces gráficas programadas en lenguaje C++. Las WxWidgets proporcionan una interfaz gráfica basada en las bibliotecas ya existentes en el sistema (nativas), con lo que se integran de forma óptima y resultan muy portables entre distintos sistemas operativos.

4.2. Análisis del Sistema

El propósito de este proceso es conseguir la especificación detallada del sistema de información a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso de Diseño del Sistema de Información.

Como ya hemos comentado MÉTRICA Versión 3 cubre tanto desarrollos estructurados como orientados a objetos y las actividades de ambas aproximaciones están integradas en una estructura común aunque presenta alguna actividad exclusiva para cada tipo de desarrollo.

Para el desarrollo de un proyecto es muy importante realizar un modelado tanto de análisis y diseño para obtener un conjunto de modelos que nos permitan especificar los requisitos, la estructura y el comportamiento del sistema.

Comenzaremos nuestro análisis del sistema realizando el modelado de casos de uso. Entendemos un caso de uso como una descripcón que especifica un comportamiento deseado del sistema. Los casos de uso describen qué hace el sistema representando los requisitos funcionales.

Describiremos los casos de uso de "Calendarios" ayudándonos de representaciones gráficas, diagramas de casos de uso y de una descripción del mismo.

Los casos de uso son una técnica de especificación de requisitos válida tanto en desarrollos estructurados como en orientación a objetos, aunque en este último caso se propone como técnica obligatoria al ser necesaria como referencia a lo largo de todo el ciclo de vida. En esta tarea se elabora el modelo de casos de uso, según las normas y estándares de la organización, identificando:

- Actores.
- Casos de uso.
- Breve descripción de cada caso de uso.

Esta descripción incluirá el nombre del caso de uso, actores que intervienen en él, precondiciones, postcondiciones, escenario principal y los posibles escenarios alternativos.

Los actores del sistema, son aquellos que intervienen en la aplicación. Entendemos por escenario principal, el flujo normal que seguirá la aplicación para este caso de uso y entendemos por escenarios alternativos, aquellas variantes que se pueden dar durante el transcurso del caso de uso (errores, otras opciones, etc).

4.2.1. Modelos de Casos de Uso

Empezaremos agrupando los diferentes casos de uso y seguidamente descenderemos en abstracción hasta los casos de uso más concretos.

4.2.2. *Casos de uso asociados a los requisitos funcionales*

Los casos de uso son una técnica de especificación de requisitos válida tanto en desarrollos estructurados como en orientación a objetos, aunque en este último caso se propone como técnica obligatoria al ser necesaria como referencia a lo largo de todo el ciclo de vida. En esta tarea se elabora el modelo de casos de uso, según las normas y estándares de la organización, identificando:

- Actores.
- Casos de uso.
- Breve descripción de cada caso de uso.

Los productos obtenidos en la tarea Determinación del Alcance del Sistema, son tomados como referencia durante la obtención de requisitos, de forma que todos los requisitos especificados se encuentren dentro del ámbito del sistema de información.

4.2.3. *Diagramas de los Casos de Uso*

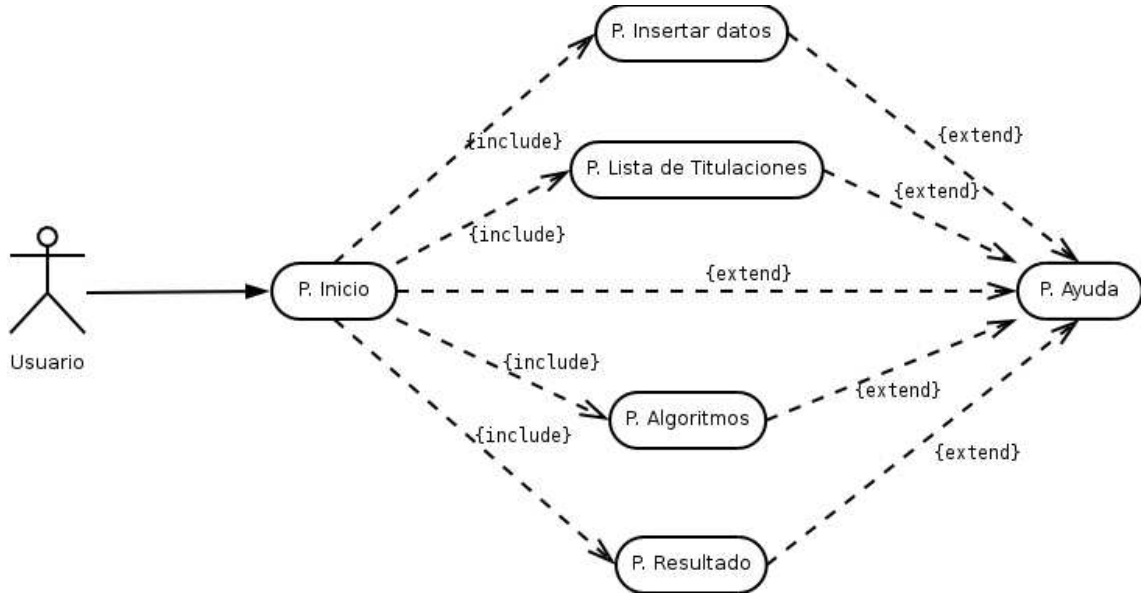

Figura 4.1: Diagrama general de los casos de uso

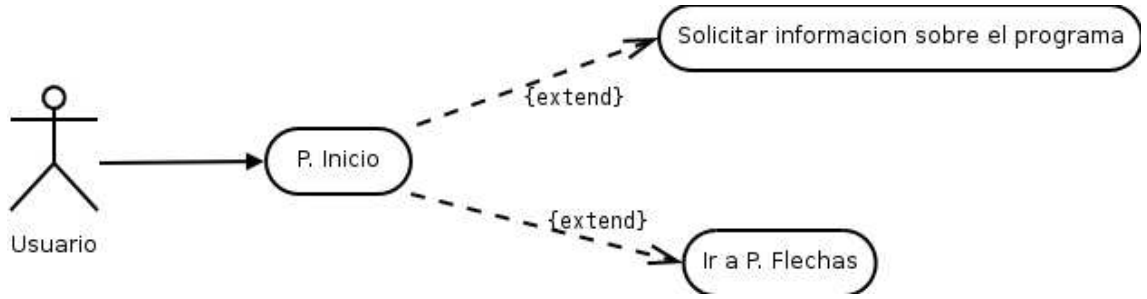

Figura 4.2: Diagrama del caso de uso PInicial, la pantalla Inicial

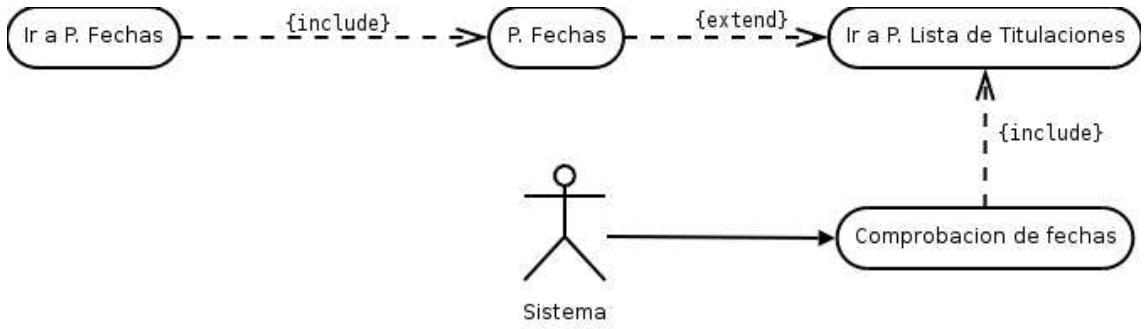

Figura 4.3: Diagrama del caso de uso PFechas, la pantalla Inserción de datos

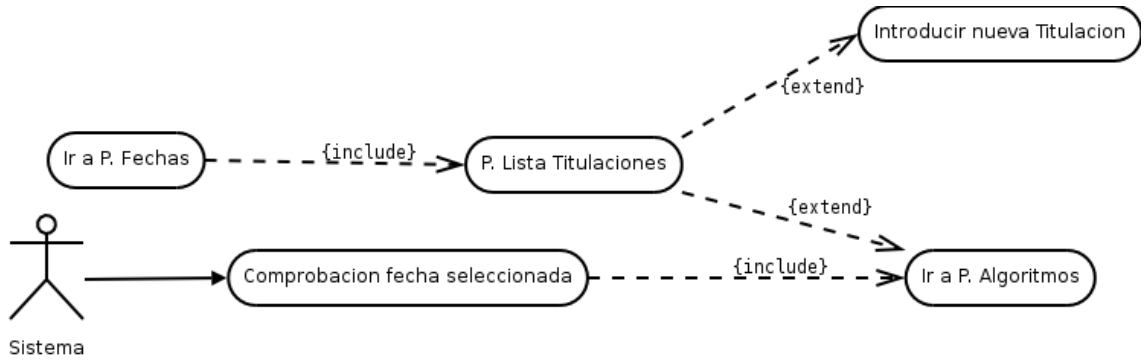

Figura 4.4: Diagrama del caso de uso PTitulaciones, la pantalla Lista de titulaciones

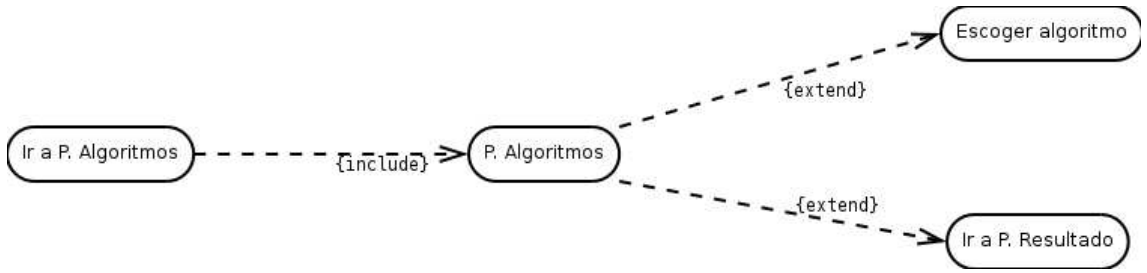

Figura 4.5: Diagrama del caso de uso PAlgoritmos, la pantalla Algoritmos

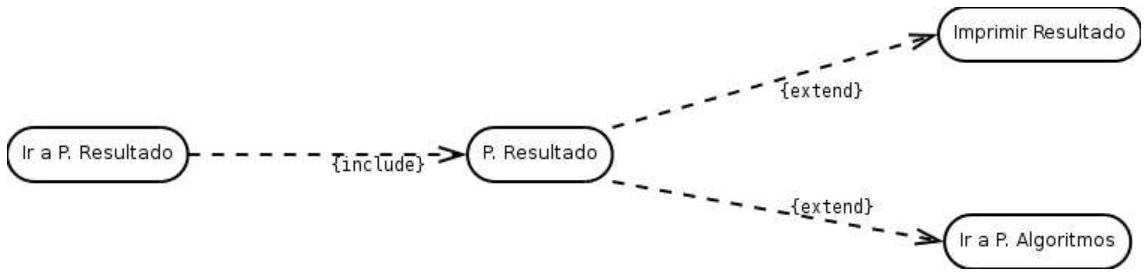

Figura 4.6: Diagrama del caso de uso PResultado, la pantalla Resultado

4.2.4. *Especificación de Casos de Uso*

Esta tarea es obligatoria en el caso de orientación a objetos y opcional en el análisis estructurado, como apoyo a la obtención de requisitos.

El objetivo de esta tarea es especificar cada caso de uso identificado en la tarea anterior, desarrollando el escenario.

Para completar los casos de uso, es preciso especificar información relativa a:

- Descripción del escenario, es decir, cómo un actor interactua con el sistema y cual es la respuesta obtenida.
- Precondiciones y poscondiciones.
- Resumen.
- Escenario principal.
- Condiciones de fallo que afectan al escenario, así como la respuesta del sistema (escenarios secundarios).

4.2.5. Caso de uso PInicial

Caso de uso: PInicial

Descripción: El sistema muestra la pantalla principal del programa y el usuario interactua en ésta.

Precondiciones: Ninguna.

Postcondiciones: Mostrará la siguiente pantalla "Pantalla inserción de datos".

Resumen: El sistema muestra la pantalla principal de la aplicación, el usuario interactua en ella dando paso a la siguiente pantalla del programa.

Escenario principal:

a) El usuario ejecuta el programa "Calendarios".

b) El sistema muestra la pantalla de inicio de la aplicación.

c) El usuario escoge continuar en la siguiente pantalla.

d) El sistema le muestra la pantalla "Inserción de datos".

Escenarios alternativos:

- *a El usuario decide abandonar el programa.
- 3a. El usuario escoge mostrar información sobre la aplicación.
  - 3a.1 El sistema le muestra un mensaje con la información del programa.
- 3b. El usuario solicita la ayuda del programa.
  - 3b.1 El sistema le muestra la pantalla "Ayuda" de la aplicación por la página inicial de la ayuda.

4.2.6. Caso de uso PFecha

Caso de uso: PFecha

Descripción: El sistema muestra la pantalla Fecha del programa y el usuario interactua en ésta.

Precondiciones: Ninguna.

Postcondiciones: Mostrará la siguiente pantalla "Pantalla lista de titulaciones".

Resumen: El sistema muestra la pantalla Fecha de la aplicación, el usuario interactua en ella dando paso a la siguiente pantalla del programa.

Escenario principal:

a) El usuario introduce la fecha de inicio del rango del calendario de exámenes.

b) El usuario introduce la fecha de finalización del rango del calendario de exámenes.

c) El usuario escoge continuar con la siguiente pantalla.

d) El sistema comprueba que la fecha de fin sea mayor que la fecha de inicio y le muestra la pantalla "Lista de Titulaciones".

Escenarios alternativos:

- *a El usuario decide abandonar el programa.
- 3a. El usuario solicita la ayuda del programa.
  - 3a.1 El sistema muestra la pantalla "Ayuda" de la aplicación por la página de ayuda de la inserción de fechas.
- 4a. El usuario ha introducido las fechas de forma incorrecta.
  - 4a.1 El sistema se lo indica y le pide al usuario que vuelva a introducirlas de forma correcta.

4.2.7. Caso de uso PLista Titulaciones

Caso de uso: PLista Titulaciones

Descripción: El sistema muestra la pantalla Lista Titulaciones del programa y el usuario interactua en ésta.

Precondiciones: Las fechas del rango del calendario de exámenes tiene que ser correcta (Fecha fin mayor que fecha inicio).

Postcondiciones: Mostrará la siguiente pantalla "Pantalla Algoritmos".

Resumen: El sistema muestra la pantalla Lista Titulaciones de la aplicación, el usuario interactua en ella dando paso a la siguiente pantalla del programa.

Escenario principal:

a) El usuario selecciona la titulación de la que quiere realizar el calendario de exámenes.

b) El usuario decide continuar con la siguiente pantalla.

c) El sistema comprueba si hay una titulación seleccionada y le muestra la pantalla "Algoritmos".

Escenarios alternativos:

- *a El usuario decide abandonar el programa.
- 2a. El usuario no encuentra la titulación y decide buscarla.
  - 2a.1 El usuario introduce el código de la titulación.
  - 2a.2 El sistema lee el código y compara con cada una de las titulaciones. Si la titulación está, el sistema la señala en la lista de titulaciones. Si la titulación no está, el sistema le indica al usuario que la introduzca.
- 2b. El usuario quiere introducir una nueva titulación.
  - 2b.1 El sistema le muestra la pantalla "Nueva" de la aplicación.
- 2c. El usuario solicita la ayuda del programa.
  - 2c.1 El sistema le muestra la pantalla "Ayuda" de la aplicación por la página de Lista Titulaciones.
- 3a. El usuario no ha seleccionado una titulación.
  - 3a.1 El sistema se lo indica mostrándole un mensaje de advertencia y le muestra nuevamente la lista de titulaciones.

4.2.8. Caso de uso PNueva

Caso de uso: PNueva

Descripción: El sistema muestra la pantalla Nueva del programa y el usuario interactua en ésta.

Precondiciones: Ninguna.

Postcondiciones: Se almacena una nueva titulación que se verá reflejada en la pantalla de Lista de Titulaciones.

Resumen: El sistema muestra la pantalla Nueva de la aplicación, el usuario interactua en ella quedando la nueva titulación reflejada en la lista de la pantalla Lista de Titulaciones.

Escenario principal:

a) El usuario introduce el código de la titulación.

b) El usuario introduce el nombre de la titulación.

c) El usuario decide guardar la nueva titulación.

Escenarios alternativos:

- *a El usuario decide abandonar el programa.
- 3a. El usuario solicita la ayuda del programa.
  - 3a.1 El sistema le muestra la pantalla "Ayuda" de la aplicación por la página de Algoritmos.
- 3.b El usuario decide no guardar la titulación.
  - 3b.1 El sistema le muestra la pantalla "Lista de Titulaciones" del programa sin ninguna modificación.

4.2.9. Caso de uso PAlgoritmos

Caso de uso: PAlgoritmos

Descripción: El sistema muestra la pantalla Algoritmos del programa y el usuario interactua en ésta.

Precondiciones: Se ha seleccionado una titulación de la que vamos a realizar el calendario de exámenes.

Postcondiciones: Mostrará la siguiente pantalla "Pantalla Resultado".

Resumen: El sistema muestra la pantalla Algoritmos de la aplicación, el usuario interactua en ella dando paso a la siguiente pantalla del programa.

Escenario principal:

a) El usuario selecciona el algoritmo deseado para realizar el calendario.

b) El usuario decide continuar con la siguiente pantalla.

c) El sistema le muestra la pantalla "Resultado".

Escenarios alternativos:

- *a El usuario decide abandonar el programa.
- 2a. El usuario solicita la ayuda del programa.
  - 2a.1 El sistema le muestra la pantalla "Ayuda" de la aplicación por la página de Algoritmos.

4.2.10. Caso de uso PResultado

Caso de uso: PResultado

Descripción: El sistema muestra la pantalla Resultado del programa y el usuario interactua en ésta.

Precondiciones: Ninguna.

Postcondiciones: Ninguna.

Resumen: El sistema muestra la pantalla Resultado de la aplicación con el calendario final de los exámenes, el usuario interactua en ella.

Escenario principal:

a) El sistema le muestra el calendario resultante de la titulación escogida, después de aplicar el algoritmo deseado en el rango de días introducido por el usuario.

Escenarios alternativos:

- *a El usuario decide abandonar el programa.
- 1a. El usuario solicita la ayuda del programa.
  - 1a.1 El sistema le muestra la pantalla "Ayuda" de la aplicación por la página de Algoritmos.
- 1.b El usuario decide volver a la pantalla "Algoritmos"
  - 1b.1 El sistema le muestra la pantalla "Algoritmos" del programa permitiéndole escoger un nuevo algoritmo para realizar el calendario.
- 1c. El usuario decide imprimir el resultado, el calendario.
  - 1c.1 El sistema le indica que el resultado quedará impreso en una carpeta "Resultado" dentro del sistema, indicándole su ubicación.

4.2.11. Caso de uso PAyuda

Caso de uso: PAyuda

Descripción: El sistema muestra la pantalla Ayuda del programa y el usuario interactua en ésta.

Precondiciones: Ninguna.

Postcondiciones: Ninguna.

Resumen: El sistema muestra la pantalla Ayuda de la aplicación dependiendo de la pantalla en la que nos situemos en el programa o dependiendo de la posición en que nos situemos dentro de esta pantalla.

Escenario principal:

a) El sistema le muestra la pantalla de ayuda al usuario.

Escenarios alternativos:

- 1a. El usuario solicita ir al inicio de la ayuda del programa.
  - 1a.1 El sistema le muestra la pantalla "Ayuda" de la aplicación en la página inicial.
- 1b. El usuario solicita ir al final de la ayuda del programa.
  - 1b.1 El sistema le muestra la pantalla "Ayuda" de la aplicación en la página final.
- 1c. El usuario solicita ir a la página siguiente a la actual de la ayuda.
  - 1c.1 El sistema le muestra la pantalla "Ayuda" de la aplicación en la página siguiente a la actual.
- 1d. El usuario solicita ir a la página anterior a la actual de la ayuda.
  - 1d.1 El sistema le muestra la pantalla "Ayuda" de la aplicación en la página anterior a la actual.
- 1e. El usuario solicita cerrar la ayuda.
  - 1e.1 El sistema cierra la pantalla "Ayuda" de la aplicación.

4.2.12. Modelo conceptual de datos

El objetivo de esta frase es la identificación de las necesidades de información de cada uno de los procesos que conforman el sistema de información con el fin de obtener un modelo de datos que contemple todas las entidades, relaciones y atributos necesarios para dar respuesta a dichas necesidades.

Se ha usado el modelo de datos de: Diagrama de clases. Este modelado incluye: clases, asociaciones, atributos, restricciones de seguridad... Se trata de uno de los modelos más comunes en el modelado de sistemas orientados a objetos. Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones y sus relaciones entre ellos.

Las clases representan los bloques de construcción más importantes de cualquier sistema orientado a objetos. Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

Esta notación es independiente de cualquier lenguaje de programación. El primer bloque representa el nombre de la clase, el segundo bloque contiene los atributos y el tercer bloque contiene las operaciones.

No es necesario mostrar todas las características. A veces las clases tienen tantas características, que no es conveniente mostrarlas todas. En estos casos también se pueden organizar las características usando estereotipos.

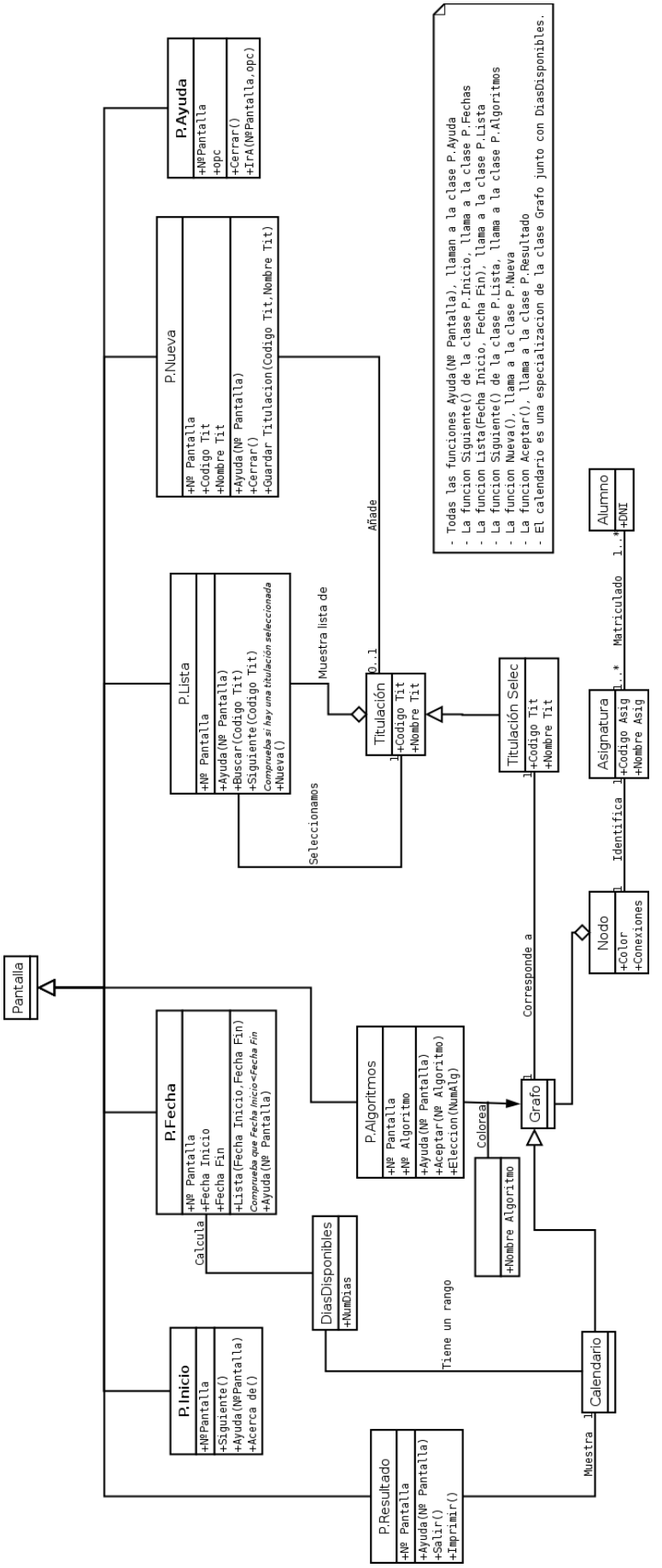

Figura 4.7: Modelo Conceptual de Datos

4.2.13. Modelo de comportamiento del sistema

El modelado del comportamiento del sistema nos permite obtener la especificación del comportamiento. Las técnicas que emplearemos para el modelado serán los diagramas de secuencia y los contratos de las operaciones.

Los diagramas de secuencia muestran la secuencia de eventos que producen los actores del sistema. Nos ayudan a la identificación de las operaciones del sistema. Los contratos de las operaciones nos permiten describir el efecto de las operaciones del sistema.

4.2.14. *Diagramas de secuencia de sistemas y Contratos de las operaciones del sistema*

Caso de uso: PInicial

Diagrama de secuencia

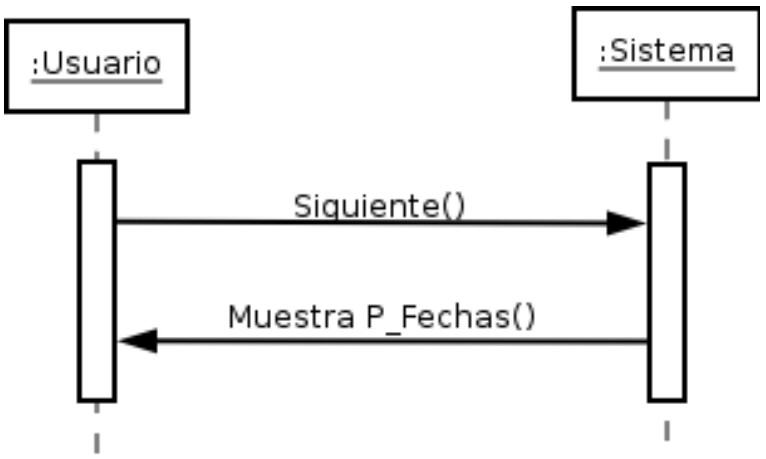

Figura 4.8: Diagrama de Comportamiento PInicial

Contratos de las operaciones

Operación: Siguiente()
Responsabilidades: Solicita ir a la siguiente pantalla (PFechas)
Precondiciones: Ninguna
Postcondiciones:

- Respuesta del sistema.

Operación: Muestra P_Fechas().
Responsabilidades: Mostrar pantalla PFechas.
Precondiciones: El usuario solicita mostrar la pantalla PFechas.
Postcondiciones:

- Muestra la pantalla PFechas.

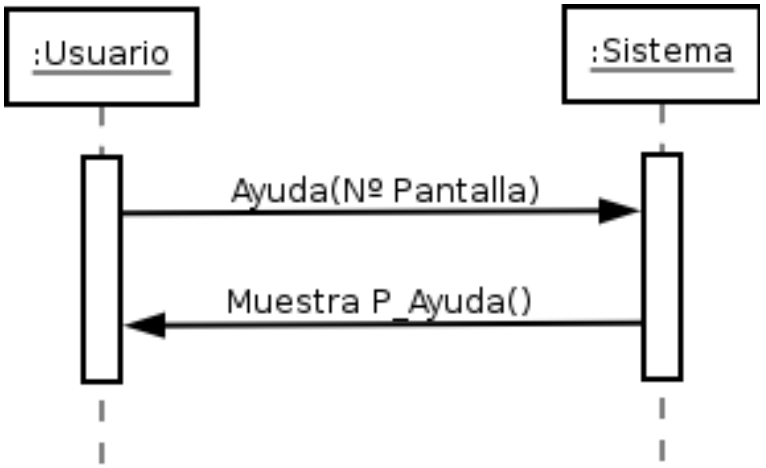

Figura 4.9: Diagrama de Comportamiento Llamada Ayuda

Contratos de las operaciones

Operación: Ayuda(Nº Pantalla)
Responsabilidades: Solicita ayuda sobre la pantalla en la que se encuentra.
Precondiciones: Ninguna
Postcondiciones:

- El sistema capta el número de pantalla.

Operación: Muestra P_Ayuda().
Responsabilidades: Mostrar pantalla PAyuda por la página correspondiente a la pantalla en la que nos situemos.
Precondiciones: El usuario solicita mostrar la ayuda de la pantalla.
Postcondiciones:

- Muestra la pantalla PAyuda con la información solicitada.

Caso de uso: PFecha

Diagrama de secuencia

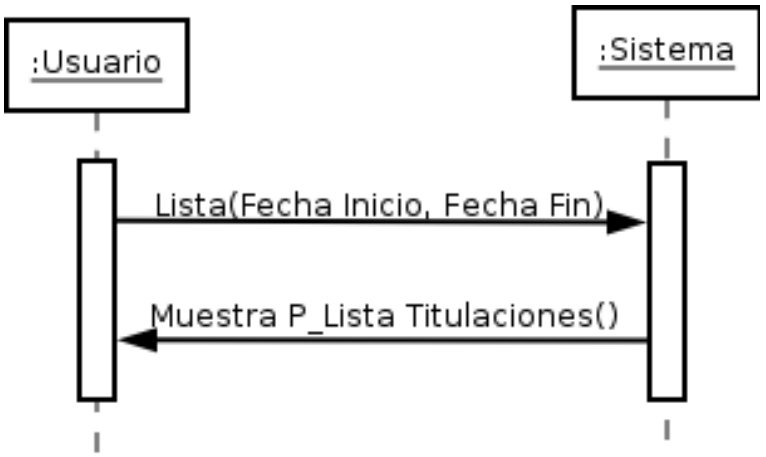

Figura 4.10: Diagrama de Comportamiento PFecha

Contratos de las operaciones

Operación: Lista(Fecha Ini, Fecha Fin)
Responsabilidades: Comprueba que Fecha Ini ¡Fecha Fin
Precondiciones: El usuario solicita visualizar la lista de las Titulaciones
Postcondiciones:

- En el caso que Fecha Ini no sea menor que Fecha Fin mostrará un mensaje de advertencia.
- Si es menor, podremos pasar a la siguiente pantalla.

Operación: Muestra P_ListaTitulaciones.
Responsabilidades: Mostrar pantalla PLista Titulaciones.
Precondiciones: El usuario solicita mostrar la pantalla PLista Titulaciones y Fecha Ini es menor que Fecha Fin.
Postcondiciones:

- Muestra la pantalla PLista Titulaciones.

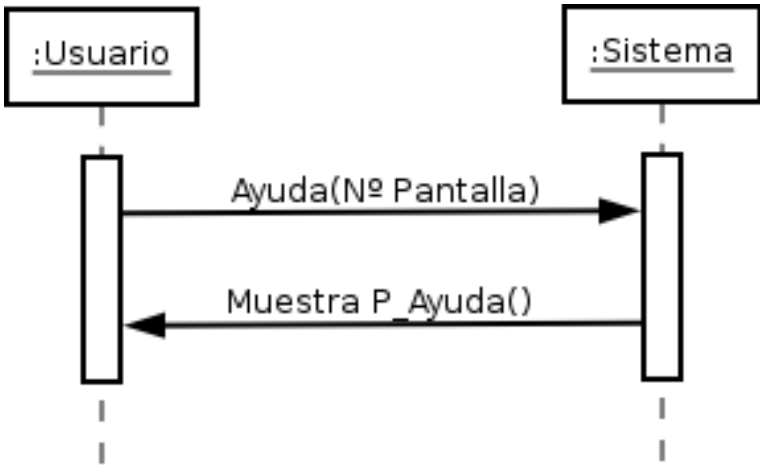

Figura 4.11: Diagrama de Comportamiento Llamada Ayuda

Contratos de las operaciones

Operación: Ayuda(Nº Pantalla)
Responsabilidades: Solicita ayuda sobre la pantalla en la que se encuentra.
Precondiciones: Ninguna
Postcondiciones:

- El sistema capta el número de pantalla.

Operación: Muestra P_Ayuda().
Responsabilidades: Mostrar pantalla PAyuda por la página correspondiente a la pantalla en la que nos situemos.
Precondiciones: El usuario solicita mostrar la ayuda de la pantalla.
Postcondiciones:

- Muestra la pantalla PAyuda con la información solicitada.

Caso de uso: PLista

Diagramas de secuencia

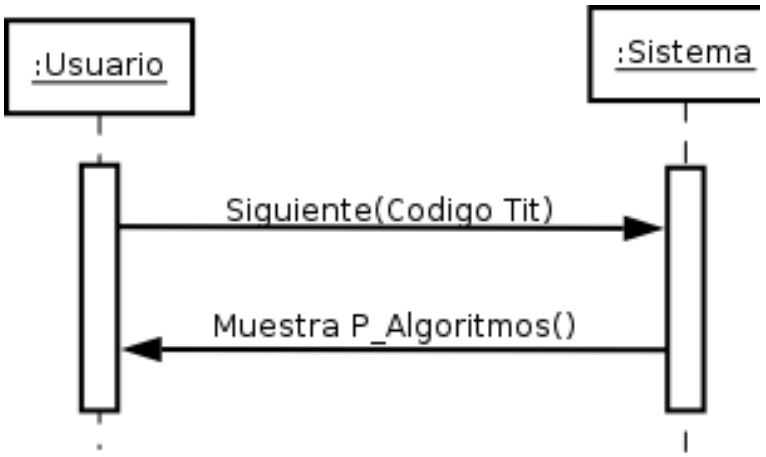

Figura 4.12: Diagrama de Comportamiento PLista, operación Siguiente

Contratos de las operaciones

Operación: Siguiente(Codigo Tit)
Responsabilidades: Comprueba que hay una Titulación seleccionada.
Precondiciones: El usuario solicita visualizar los algoritmos que ofrece la aplicación.
Postcondiciones:

- En el caso que no haya ninguna Titulación seleccionada mostrará un mensaje de advertencia.
- Si hay una Titulación seleccionada, podremos pasar a la siguiente pantalla.

Operación: Muestra P_Algoritmos.
Responsabilidades: Mostrar pantalla PAlgoritmos
. *Precondiciones:* El usuario solicita mostrar la pantalla PAlgoritmos y hay una Titulación seleccionada.
Postcondiciones:

- Muestra la pantalla PAlgoritmos.

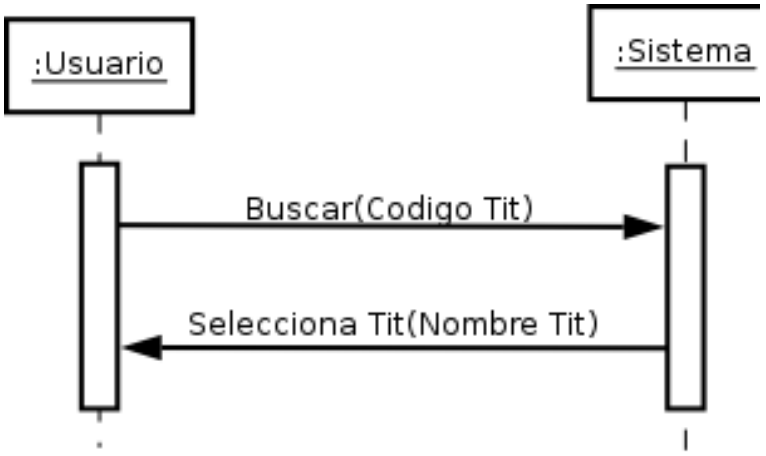

Figura 4.13: Diagrama de Comportamiento PLista, operación Buscar

Contratos de las operaciones

Operación: Buscar(Codigo Tit)
Responsabilidades: Comprueba que la Titulación se encuentra en nuestra lista de Titulaciones y muestra en pantalla, en caso de no estar, un mensaje de advertencia y en caso de estar la señala dentro de nuestra lista.
Precondiciones: El usuario solicita buscar una Titulación en nuestra lista.
Postcondiciones:

- En el caso que no exista dicha Titulación mostrará un mensaje de advertencia, invitando al usuario que añada dicha Titulación.
- Si existe la Titulación, podremos pasar a la siguiente pantalla.

Operación: Selecciona Tit(Nombre Tit)
. *Responsabilidades:* Seleccionar en la pantalla, en caso de existir, la Titulación en la lista, en caso contrario mostra mensaje de advertencia.
Precondiciones: El usuario quiere buscar una Titulación e introduce el código de la misma.
Postcondiciones:

- Muestra un mensaje de advertencia en caso de no existir.
- Señala en la lista de Titulaciones la Titulación buscada.

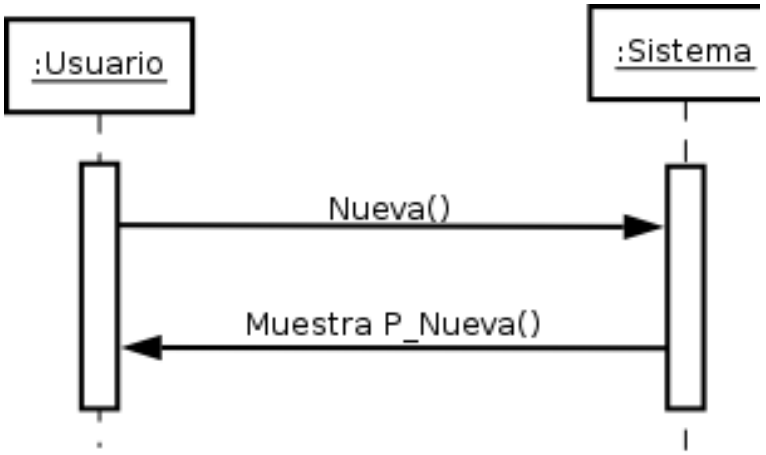

Figura 4.14: Diagrama de Comportamiento PLista, operación Nueva

Contratos de las operaciones

Operación: Nueva()
Responsabilidades: Solicita ir a la pantalla PNueva.
Precondiciones: Ninguna.
Postcondiciones:

- Respuesta del sistema.

Operación: Muestra P_Nueva().
Responsabilidades: Mostrar pantalla PNueva.
Precondiciones: El usuario solicita mostrar la pantalla PNueva.
Postcondiciones:

- Muestra la pantalla PNueva.

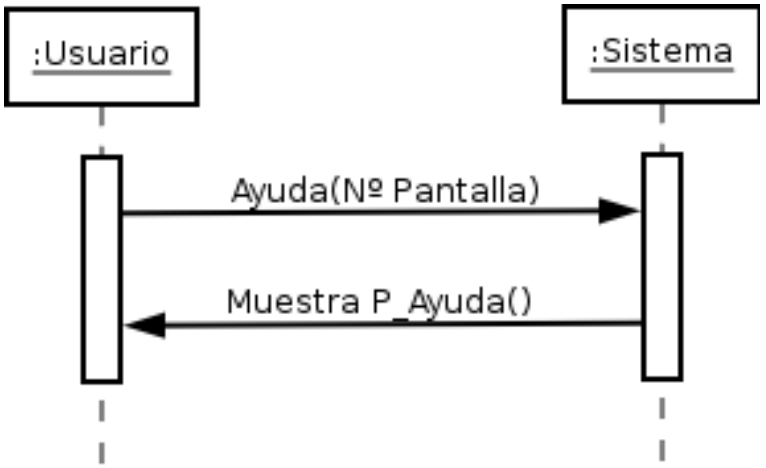

Figura 4.15: Diagrama de Comportamiento Llamada Ayuda

Contratos de las operaciones

Operación: Ayuda(Nº Pantalla)
Responsabilidades: Solicita ayuda sobre la pantalla en la que se encuentra.
Precondiciones: Ninguna
Postcondiciones:

- El sistema capta el número de pantalla.

Operación: Muestra P_Ayuda().
Responsabilidades: Mostrar pantalla PAyuda por la página correspondiente a la pantalla en la que nos situemos.
Precondiciones: El usuario solicita mostrar la ayuda de la pantalla.
Postcondiciones:

- Muestra la pantalla PAyuda con la información solicitada.

Caso de uso: PNueva

Diagramas de secuencia

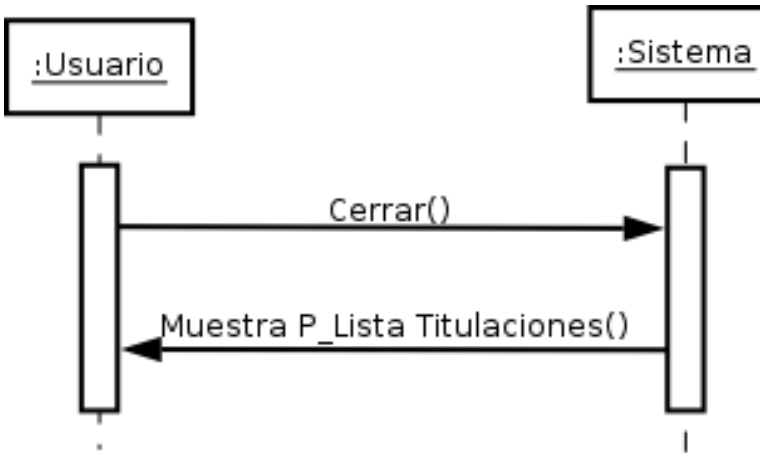

Figura 4.16: Diagrama de Comportamiento PNueva, operación Cerrar

Contratos de las operaciones

Operación: Cerrar().
Responsabilidades: Cerrar la pantalla PNueva sin realizar ningún cambio en la lista de Titulaciones.
Precondiciones: El usuario después de entrar en la pantalla PNueva, no quiere introducir ninguna Titulacion.
Postcondiciones:

- Respuesta del sistema.

Operación: Muestra P_Lista Titulaciones().
Responsabilidades: Mostrar pantalla PLista Titulaciones.
Precondiciones: El usuario solicita cerrar la pantalla PNueva porque no quiere introducir una nueva Titulación.
Postcondiciones:

- Muestra la pantalla PLista Titulaciones.

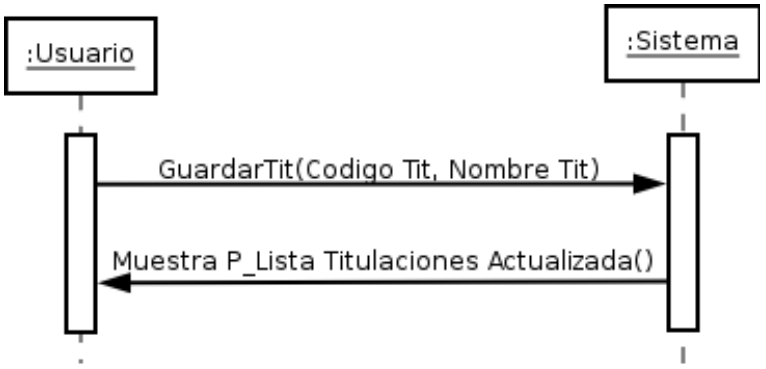

Figura 4.17: Diagrama de Comportamiento PNueva, operación Guardar

Contratos de las operaciones

Operación: GuardarTit(Codigo Tit, Nombre Tit)
Responsabilidades: Almacenar en la lista de Titulaciones la nueva carrera con los datos Codigo Tit y Nombre Tit.
Precondiciones: El usuario introduce correctamente los datos de la nueva Titulación.
Postcondiciones:

- Se guarda una Titulación nueva en la lista donde su código es Codigo Tit y su nombre es Nombre Tit.

Operación: Muestra P_Lista Titulaciones Actualizada().
Responsabilidades: Mostrar pantalla PLista Titulaciones actualizada, con la carrera nueva.
Precondiciones: El usuario ha introducido los datos de la nueva Titulación correctamente.
Postcondiciones:

- Muestra la pantalla PLista Titulaciones con la nueva Titulación introducida por el usuario, lista actualizada.

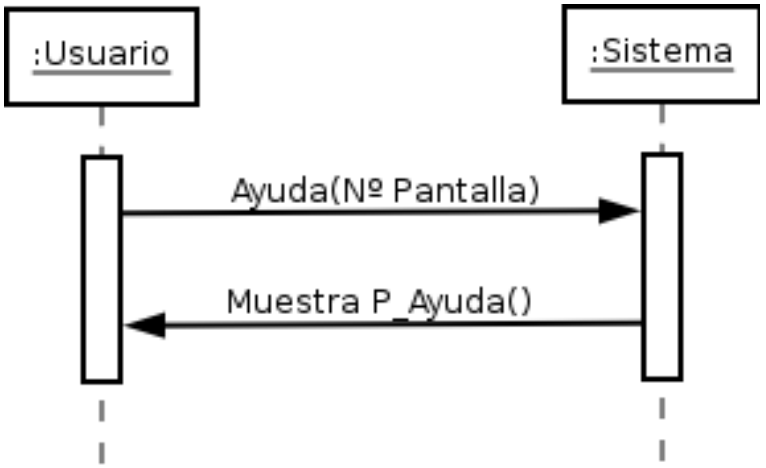

Figura 4.18: Diagrama de Comportamiento Llamada Ayuda

Contratos de las operaciones

Operación: Ayuda(Nº Pantalla)
Responsabilidades: Solicita ayuda sobre la pantalla en la que se encuentra.
Precondiciones: Ninguna
Postcondiciones:

- El sistema capta el número de pantalla.

Operación: Muestra P_Ayuda().
Responsabilidades: Mostrar pantalla PAyuda por la página correspondiente a la pantalla en la que nos situemos.
Precondiciones: El usuario solicita mostrar la ayuda de la pantalla.
Postcondiciones:

- Muestra la pantalla PAyuda con la información solicitada.

Caso de uso: PAlgoritmos

Diagramas de secuencia

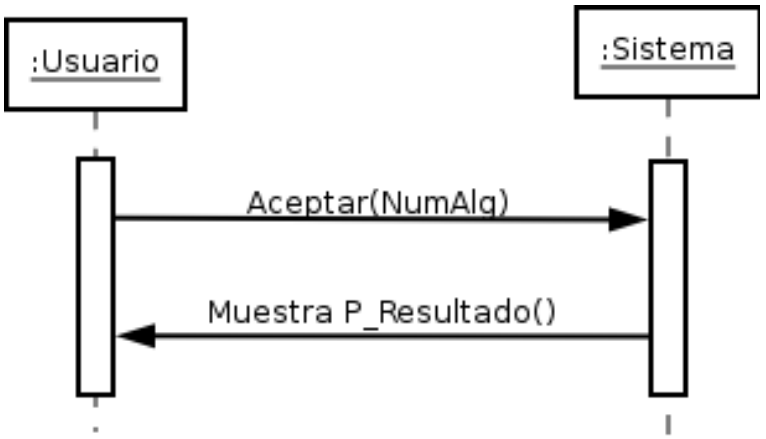

Figura 4.19: Diagrama de Comportamiento PAlgoritmos, operación Aceptar

Contratos de las operaciones

Operación: Aceptar(NumAlg)
Responsabilidades: Según el algoritmo que haya sido seleccionado "NumAlg", llamar al algoritmo y que éste realice las operaciones pertinentes.
Precondiciones: El usuario ha seleccionado un algoritmo "NumAlg", de los tres que se le muestran en pantalla.
Postcondiciones:

- Llamamos al algoritmo seleccionado "NumAlg" para que coloree "nuestro grafo".

Operación: Muestra P_Resultado().
Responsabilidades: Mostrar pantalla PResultado.
Precondiciones: El usuario solicita obtener el resultado del calendario mediante uno de los algoritmos que le proporcionamos.
Postcondiciones:

- Muestra la pantalla PResultado con el calendario resultante del algoritmo escogido.

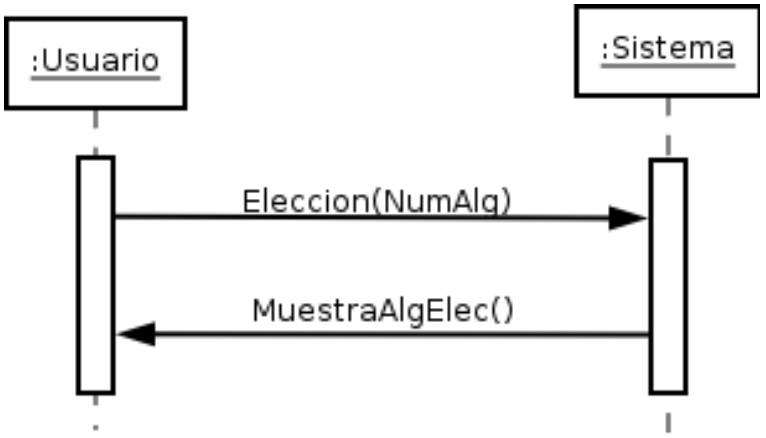

Figura 4.20: Diagrama de Comportamiento PNueva, operación Elección

Contratos de las operaciones

Operación: Eleccion(NumAlg)
Responsabilidades: Determinar el número del algoritmo que ha sido seleccionado para mostrar por pantalla o bien para realizar el calendario.
Precondiciones: El usuario solicita ver la información correspondiente al algoritmo "NumAlg".
Postcondiciones:

- El sistema recoge el número del algoritmo a mostrar y se dispone a mostrarlo.

Operación: MuestraAlgElec().
Responsabilidades: Mostrar por pantalla las diferentes características y el funcionamiento del algoritmo que ha sido seleccionado.
Precondiciones: El usuario ha solicitado ver la información y el funcionamiento de uno de los algoritmos.
Postcondiciones:

- Muestra en la pantalla PAlgoritmos la información del algoritmo escogido, bajo la petición del usuario.

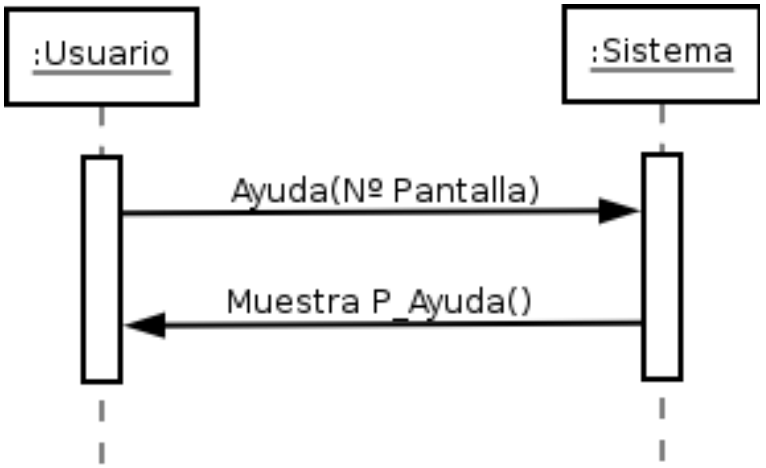

Figura 4.21: Diagrama de Comportamiento Llamada Ayuda

Contratos de las operaciones

Operación: Ayuda(Nº Pantalla)
Responsabilidades: Solicita ayuda sobre la pantalla en la que se encuentra.
Precondiciones: Ninguna
Postcondiciones:

- El sistema capta el número de pantalla.

Operación: Muestra P_Ayuda().
Responsabilidades: Mostrar pantalla PAyuda por la página correspondiente a la pantalla en la que nos situemos.
Precondiciones: El usuario solicita mostrar la ayuda de la pantalla.
Postcondiciones:

- Muestra la pantalla PAyuda con la información solicitada.

Caso de uso: PResultado

Diagramas de secuencia

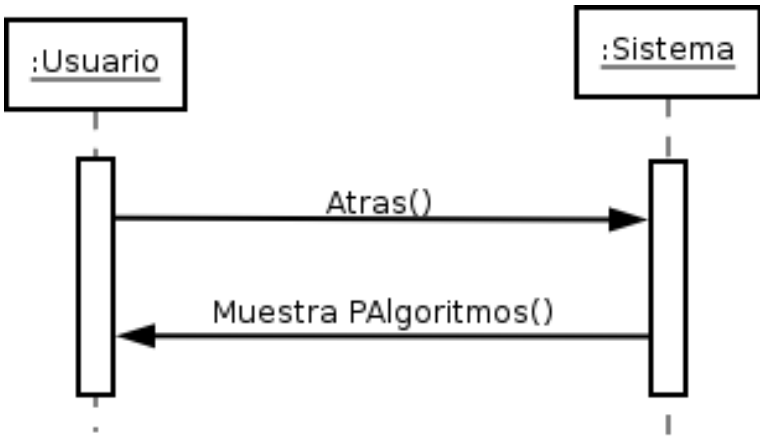

Figura 4.22: Diagrama de Comportamiento PResultado, operación Atrás

Contratos de las operaciones

Operación: Atras()
Responsabilidades: Devolver al usuario a la pantalla de PAlgoritmos para que pueda seleccionar otro algoritmo con el que realizar el calendario.
Precondiciones: El usuario ha decidido volver hacia atrás y ha pulsado el botón "Atrás".
Postcondiciones:

- El sistema se dispone a cerrar la pantalla PResultado y a mostrar la pantalla PAlgoritmos

Operación: Muestra PAlgoritmos().
Responsabilidades: Mostrar pantalla PAlgoritmos.
Precondiciones: El usuario solicita volver a la pantalla PAlgoritmos para obtener el resultado del calendario mediante uno de los algoritmos que le proporcionamos.
Postcondiciones:

- Muestra la pantalla PAlgoritmos.

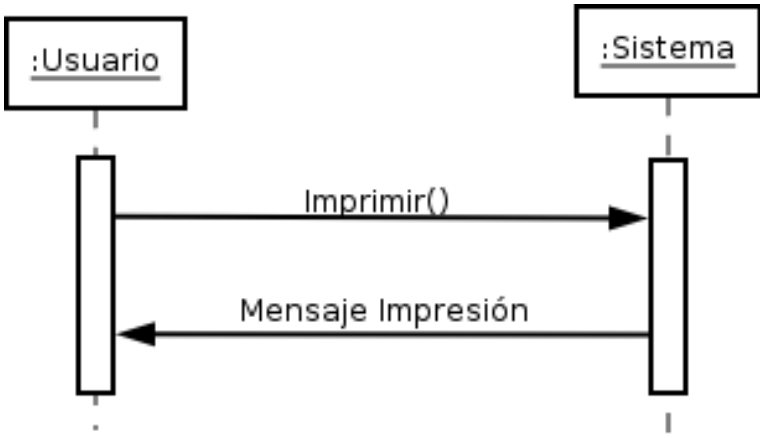

Figura 4.23: Diagrama de Comportamiento PResultado, operación Imprimir

Contratos de las operaciones

Operación: Imprimir()
Responsabilidades: Imprimir el resultado en un PDF que generamos y almacenarlo en el directorio que le indicamos al usuario.
Precondiciones: El usuario ha decidido que quiere imprimir el calendario resultante.
Postcondiciones:

- El sistema muestra un mensaje indicando la ubicación del PDF resultante y muestra por pantalla un mensaje indicándoselo al usuario.

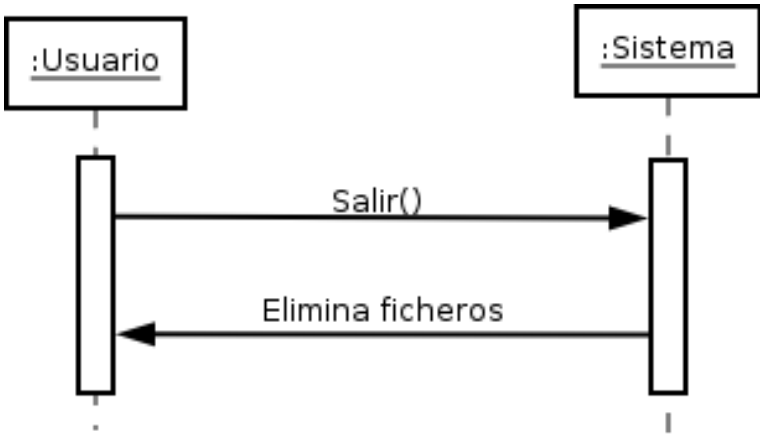

Figura 4.24: Diagrama de Comportamiento PResultado, operación Imprimir

Contratos de las operaciones

Operación: Salir()
Responsabilidades: Salir del programa eliminando todos los ficheros que ya no serán necesarios y que han sido generados a lo largo de la aplicación.
Precondiciones: El usuario decide finalizar el programa.
Postcondiciones:

- El sistema elimina los ficheros generados por la aplicación y que no serán necesarios y salimos del programa.

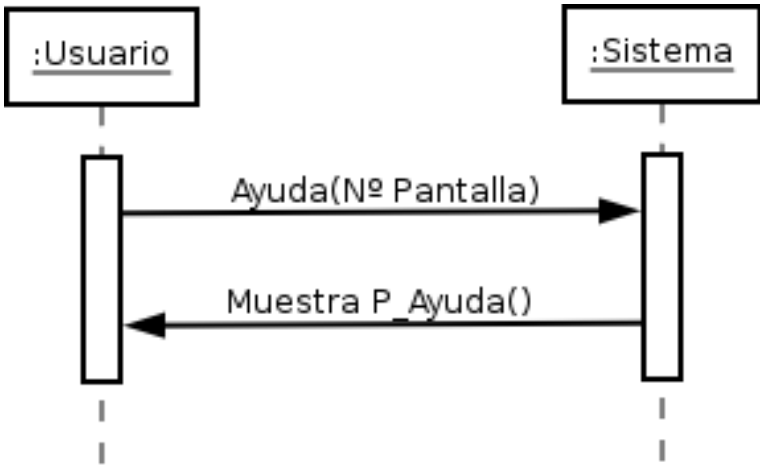

Figura 4.25: Diagrama de Comportamiento Llamada Ayuda

Contratos de las operaciones

Operación: Ayuda(Nº Pantalla)
Responsabilidades: Solicita ayuda sobre la pantalla en la que se encuentra.
Precondiciones: Ninguna
Postcondiciones:

- El sistema capta el número de pantalla.

Operación: Muestra P_Ayuda().
Responsabilidades: Mostrar pantalla PAyuda por la página correspondiente a la pantalla en la que nos situemos.
Precondiciones: El usuario solicita mostrar la ayuda de la pantalla.
Postcondiciones:

- Muestra la pantalla PAyuda con la información solicitada.

Caso de uso: PAyuda

Diagramas de secuencia

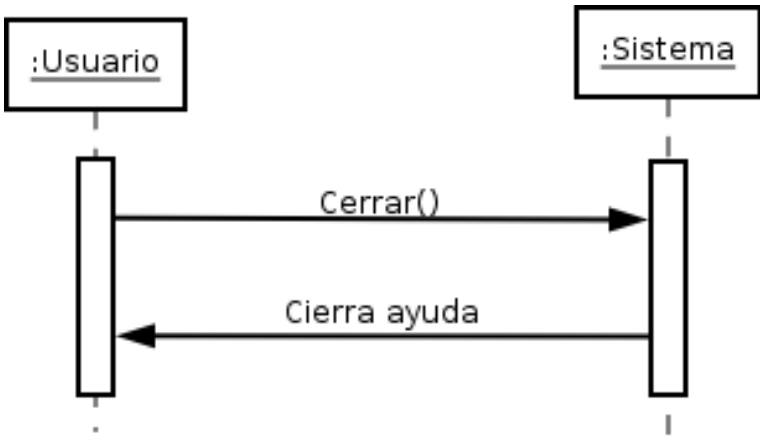

Figura 4.26: Diagrama de Comportamiento PAyuda, operación Cerrar

Contratos de las operaciones

Operación: Cerrar()
Responsabilidades: Cerrar la pantalla PAyuda.
Precondiciones: El usuario ha decidido no visualizar la ayuda del programa.
Postcondiciones:

- El sistema cierra la pantalla PAyuda.

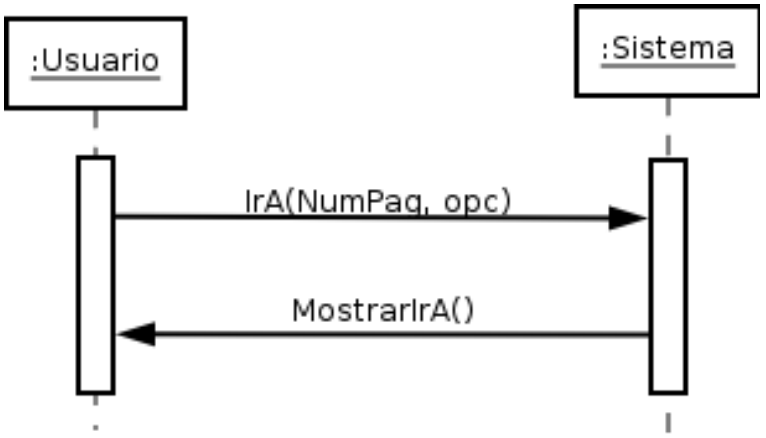

Figura 4.27: Diagrama de Comportamiento PAyuda, operación IrA

Contratos de las operaciones

Operación: IrA(NumPag, opc)
Responsabilidades: Controlar la posición en la que estemos dentro de la ayuda y la posición a la que tenemos que llegar en ella.
Precondiciones: El usuario ha decidido navegar en la ayuda desde una página dada y nos moveremos según la opción seleccionada "opc".
Postcondiciones:

- El sistema controla la posición de la ayuda y a la que queremos llegar.

Operación: MostrarIrA()
Responsabilidades: Mostrar la posición deseada, dentro de la ayuda, a la que el usuario quiere dirigirse.
Precondiciones: El usuario ha decidido navegar en la ayuda desde una página dada y nos moveremos según la opción seleccionada: Ir a Inicio, Fin, Siguiente, Anterior.
Postcondiciones:

- El sistema muestra en la pantalla PAyuda la información de la página solicitada.

4.2.15. Definición de Interfaces de Usuario

Especificamos las interfaces entre el sistema y el usuario: formatos de pantallas, diálogos e informes, principalmente. El objetivo es realizar un análisis de los procesos del sistema de información en los que se requiere una interacción del usuario, con el fin de crear una interfaz que satisfaga todos los requisitos establecidos, teniendo en cuenta los diferentes perfiles a quiénes va dirigido.

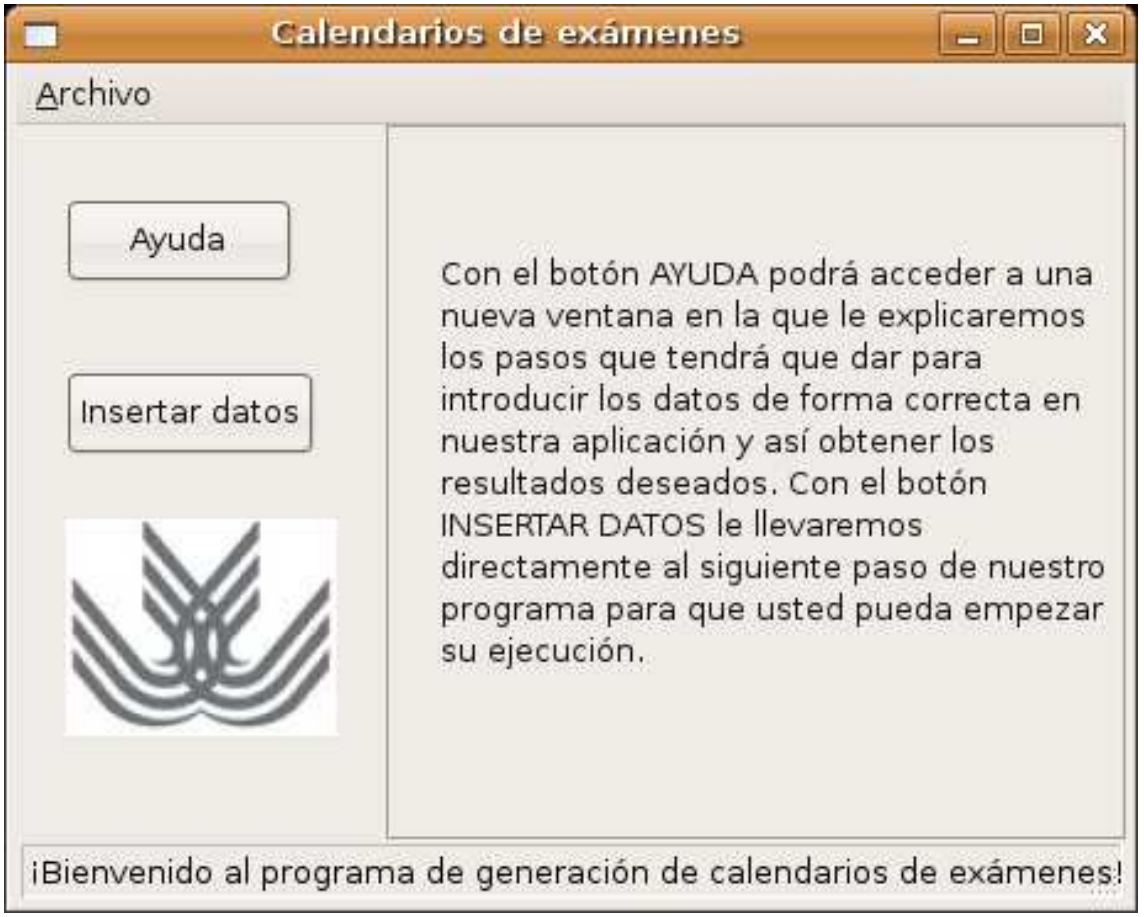

Figura 4.28: Pantalla de Inicio

Nos encontramos en la pantalla inicial, en ésta se nos dan unas breves instrucciones sobre el manejo de nuestro programa. Desde ella podemos solicitar información más detallada sobre el manejo de nuestra aplicación pulsando el botón de "Ayuda" que aparece en pantalla, o bien solicitar una breve descripción sobre el objetivo de nuestra aplicación pulsando "Archivo/Sobre...". El usuario para continuar con la aplicación pulsará "Insertar datos".

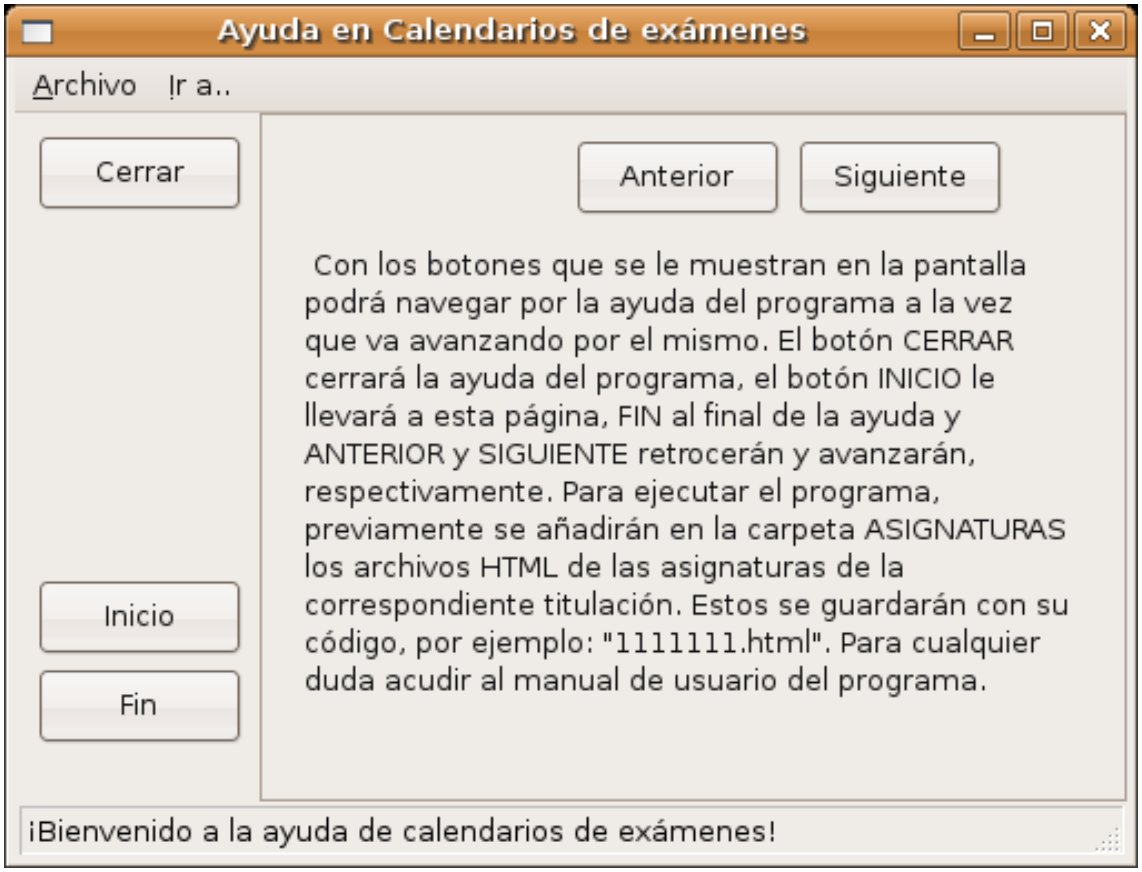

Figura 4.29: Pantalla de Ayuda

A esta pantalla podrá acceder el usuario desde cualquier punto de la aplicación, se le mostrará información según el paso de la aplicación en que nos encontremos. Obviamente podrá navegar a través de ella mediante los botones que se muestran pudiendo acceder en cualquier momento a la información deseada. Sólo tendremos que pulsar el botón “Cerrar” cuando no queramos seguir visualizando la ayuda.

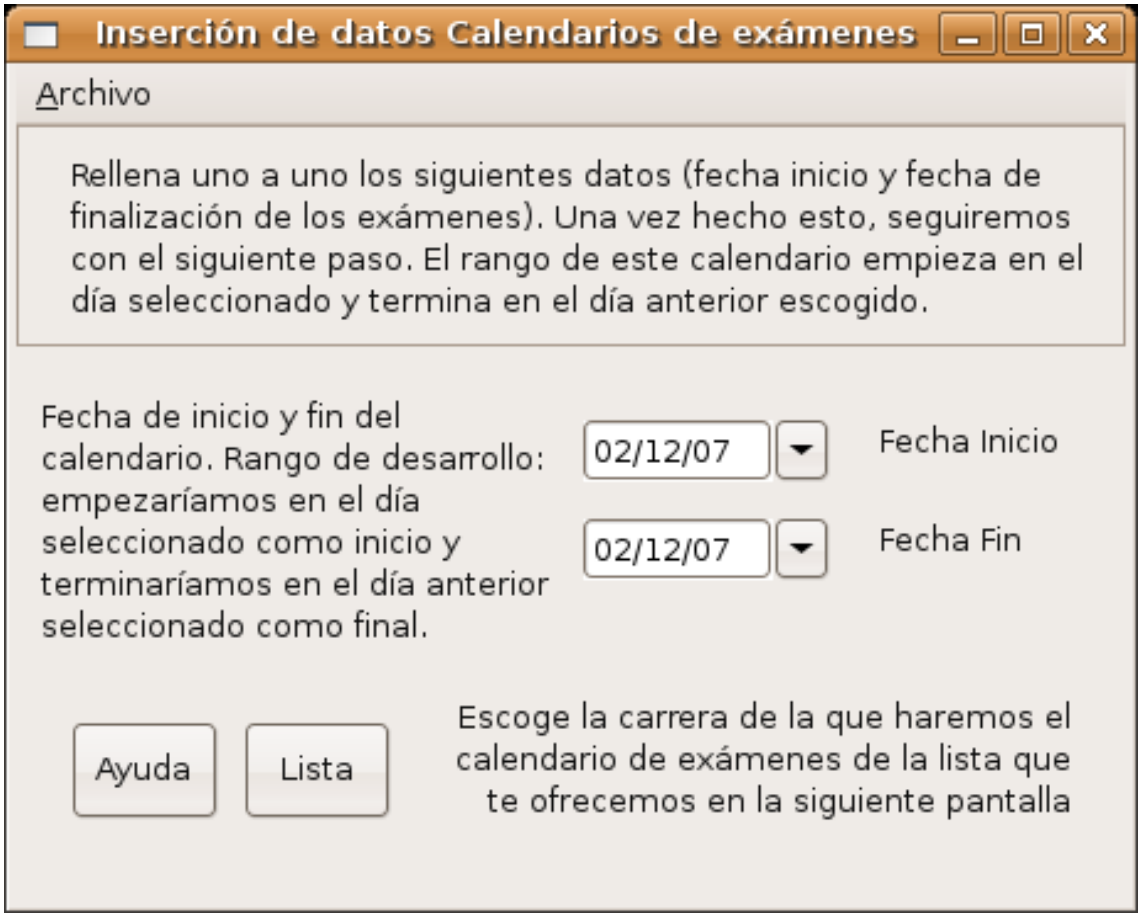

Figura 4.30: Pantalla de Insercción de datos

En esta pantalla le indicamos al usuario los pasos que tiene que dar y las indicaciones correspondientes de cada paso. Introducirá la fecha de inicio y fin del rango del calendario de exámenes. Teniendo en cuenta que el día inicial es el que se nos introduce y el día final es el día anterior al que el usuario introdujo, el último día no se cuenta. El sistema controla antes de pasar a la siguiente pantalla que las fechas sean correctas, es decir que la fecha de inicio sea menor que la fecha final del rango. Como ya hemos comentado podemos acceder siempre a la "Ayuda" donde se nos planteará de una forma más detallada los pasos a dar, respecto a esta pantalla en concreto.

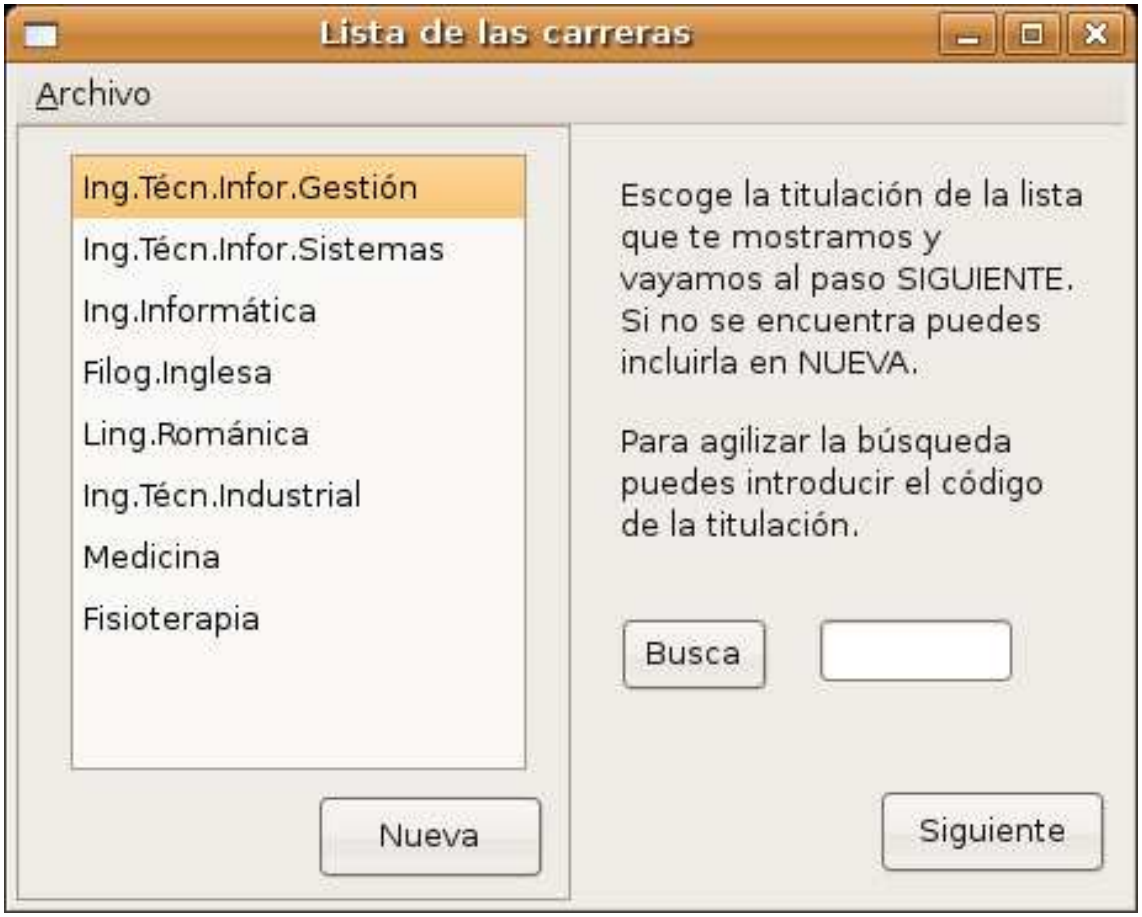

Figura 4.31: Pantalla de Lista de Titulaciones

Como podemos observar se nos muestran una lista de titulaciones en esta pantalla. Si el usuario encuentra la que va a emplear en nuestra aplicación, la seleccionará de la lista, en caso de no encontrarla o no saber el nombre y sí el código, podrá buscarla mediante la búsqueda que le ofrecemos; introduciendo el código y pulsando "Buscar" se le marcará, en el caso en que ésta esté, en la lista; en caso contrario le saldrá un mensaje indicando que no se encuentra dicha titulación y le invita a introducirla. Pulsando el botón "Nueva" el usuario podrá llevar a cabo la introducción de una nueva carrera. Y como ya hemos comentado, podrá acceder en cualquier momento a la Ayuda con la información de la correspantalla. El sistema antes de pasar a la siguiente pantalla comprobará si hay seleccionada una carrera, en caso contrario se lo indicará al usuario mostrándole un mensaje de advertencia.

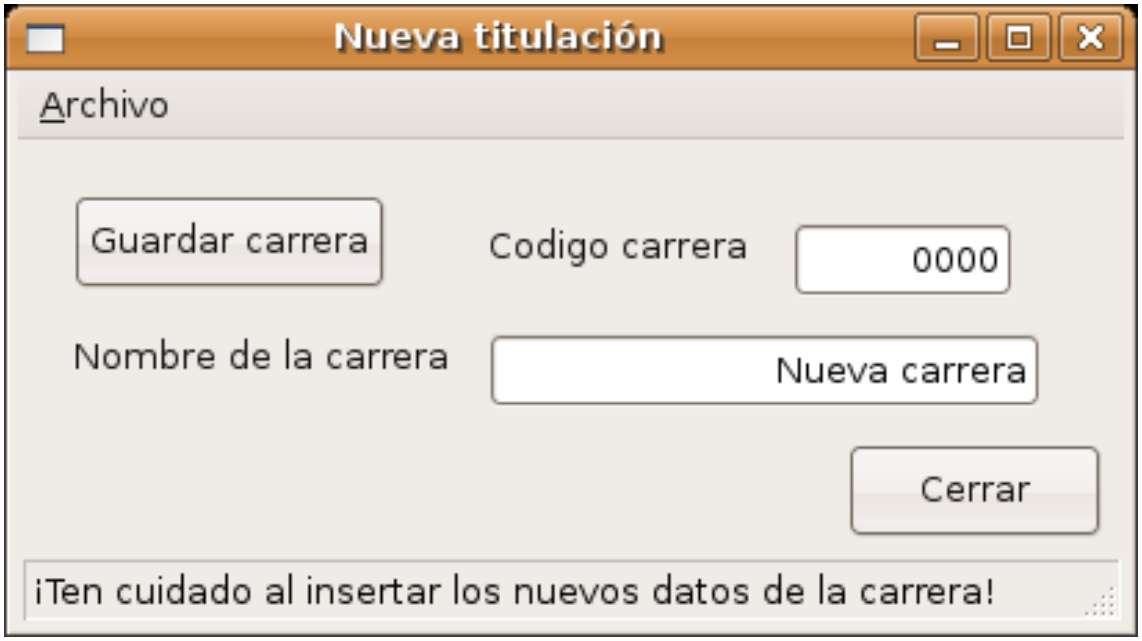

Figura 4.32: Pantalla Nueva Titulación

El usuario ha decidido introducir la nueva titulación, siempre se puede arrepentir y volver a la pantalla anterior de la "Lista de Titulaciones" sin tener que introducir ninguna carrera; en tal caso tendrá que pulsar el botón "Cerrar" y se le mostrará la pantalla sin ninguna modificación. Si quiere introducirla, previamente le habrá salido un mensaje indicándole que introduzca el nombre de la titulación sin espacios. Una vez introducidos código y nombre, pulsaremos "Guardar Carrera" (no tiene que comprobar nada el sistema ya que el código de la titulación es único) y nos aparecerá la pantalla "Lista de Titulaciones" con la nueva carrera en la lista pudiendola seleccionar sin problemas. Para mayor información sobre los pasos a dar dentro de esta pantalla, el usuario siempre podrá solicitar la ayuda del programa, apareciéndole la información necesaria para esta parte de la aplicación.

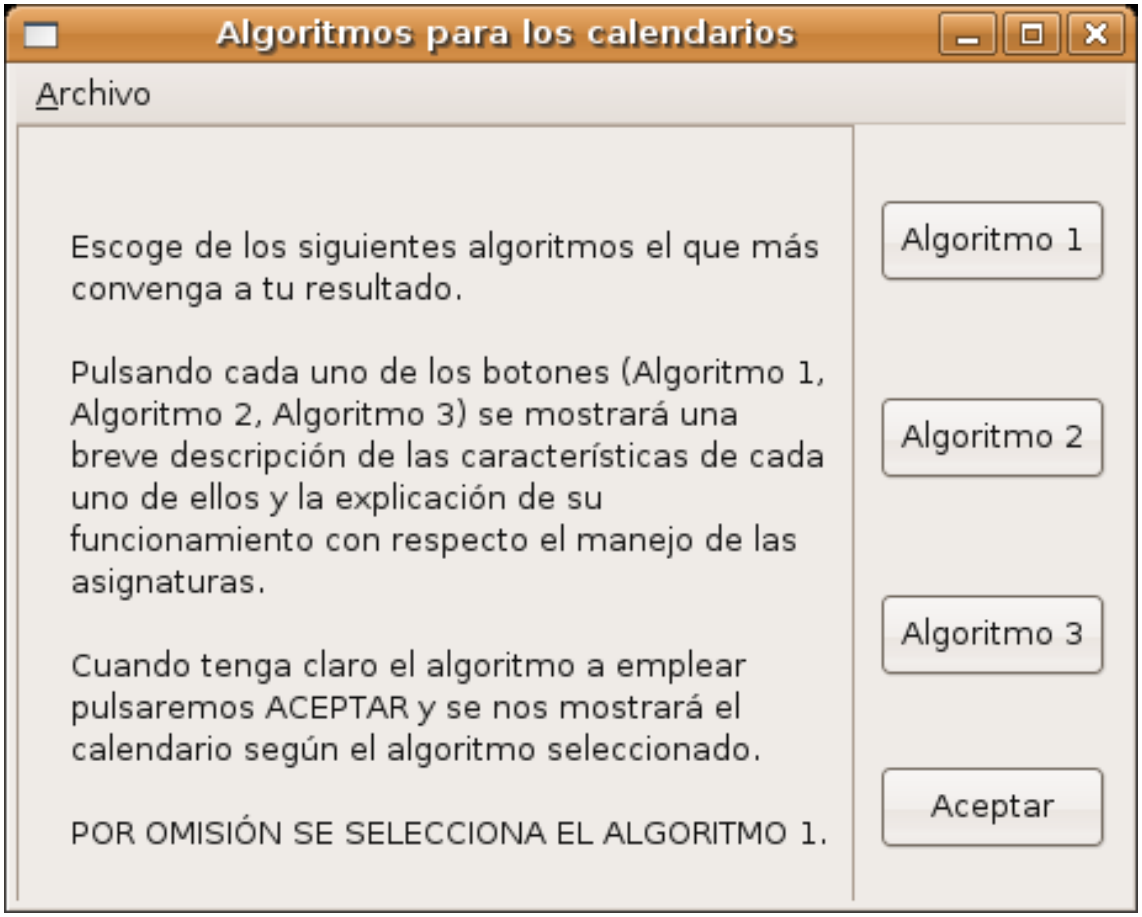

Figura 4.33: Pantalla de Algoritmos

Y antes de mostrar el resultado, volvemos a pedirle al usuario que escoja uno de los tres algoritmos que empleamos en nuestra aplicación. Inicialmente se le muestra un texto indicándole los pasos a dar, puediéndose ver de forma más detallada en la ayuda del programa. Navegando por la pantalla el usuario podrá obtener información sobre las características del algoritmo y unas breves líneas sobre el funcionamiento del mismo. Después de seleccionar el algoritmo que más se ajuste a sus necesidades pulsaremos “Aceptar” para obtener el resultado.

	A	B	C	D
1	1710043	PROGRAMACIÓN EN INTERNET	3/12/2007	10:00
2	1710032	INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL	3/12/2007	17:00
3	1710034	PROGRAMACIÓN CONCURRENTE Y DISTRIBUIDA	3/12/2007	17:00
4	1710026	TRADUCTORES	4/12/2007	10:00
5	1710031	HERRAMIENTAS DE DESARROLLO RÁPIDO DE APLICACIONES	4/12/2007	10:00
6	1710024	BASES DE DATOS I	4/12/2007	17:00
7	1710029	TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES	4/12/2007	17:00
8	1710022	PROGRAMACIÓN ORIENTADA A OBJETOS	5/12/2007	10:00
9	1710025	BASES DE DATOS II	5/12/2007	10:00
10	1710019	INTRODUCCIÓN A LA PROGRAMACIÓN	5/12/2007	17:00
11	1710021	INFORMÁTICA GENERAL	5/12/2007	17:00
12	1710017	INGENIERÍA DEL SOFTWARE I	6/12/2007	10:00
13	1710018	INGENIERÍA DEL SOFTWARE II	6/12/2007	10:00

Figura 4.34: Pantalla Resultado

Y finalmente la pantalla resultado en la que, según el algoritmo seleccionado se nos mostrará el calendario de exámenes correspondiente. La primera columna se corresponde con los códigos de las asignaturas, la siguiente con los nombres, la tercera con la fecha asignada y la cuarta con la hora. Si queremos ver otro posible resultado con otro algoritmo, el usuario tendrá que pulsar "Atrás" y se le mostrará la pantalla de Algoritmos descrita anteriormente. En el caso que quiera imprimir este resultado, sólo tendrá que pulsar el botón "Imprimir", apareciéndole un mensaje en el que se le indica la ubicación del PDF con el calendario resultado. Volver a insistir en que el usuario podrá tener acceso a la ayuda del programa en todo momento. Finalizaremos la aplicación en el botón "Salir" conservándose, siempre que escogiésemos la opción de imprimir, el resultado en PDF de la aplicación, en la ubicación indicada.

4.3. Diseño del sistema

El diseño del software forma parte del núcleo técnico del proceso de ingeniería del software y se aplica independientemente del paradigma de desarrollo utilizado. A partir del análisis y especificación de los requisistos del sofware, el diseño es la primera de las tres actividades técnicas (diseño, codificación y prueba) que hay que realizar para construir y verificar el software. Es un proceso iterativo a través del cual se traducen los requisitos en un modelo o representación del sistema que se va a construir. Inicialmente, el diseño se representa a un alto nivel de abstracción y a medida que se realizan iteraciones, el refinamiento siguiente lleva a representaciones del diseño de mucho menor nivel de abstracción.

4.3.1. Arquitectura del sistema software

En el apartado presente daremos una descripción de los subsistemas existentes y de los componentes con sus relaciones del sistema software. Dentro de la determinación de la arquitectura software necesitaremos establecer qué propiedades ha de cumplir el sistema y de qué recursos disponemos.

La arquitectura del software es la estructura del sistema, componentes, propiedades y relaciones.

En nuestra aplicación podemos encontrar una división en etapas de la arquitectura.

a) Capa de interfaz de usuario: Incluye todos los componentes utilizados para la representación de la interfaz de usuario de la aplicación.

b) Capa de control de la aplicación: Incluye componentes y procedimientos que permiten el control de eventos y mandar mensajes a las capas necesarias.

c) Capa de dominio: Incluye todas las clases con las que trabaja el sistema.

Como ya hemos comentado antes, nuestro sistema utiliza lenguaje C++ y wxWidgets que son orientados a objetos.

4.3.2. Diseño de la capa de interfaz de usuario

4.3.2.1. *Diagramas de secuencia*

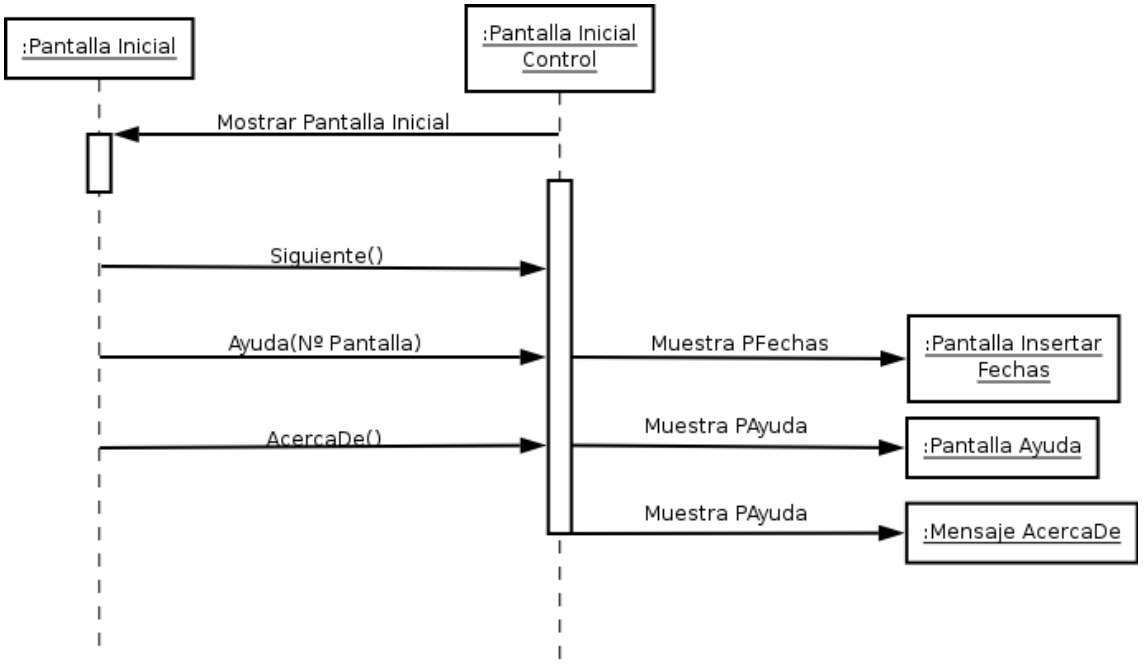

Figura 4.35: Diagrama de Interacción PInicial

El usuario cuando entra en el programa, se encuentra que puede realizar las siguientes órdenes.

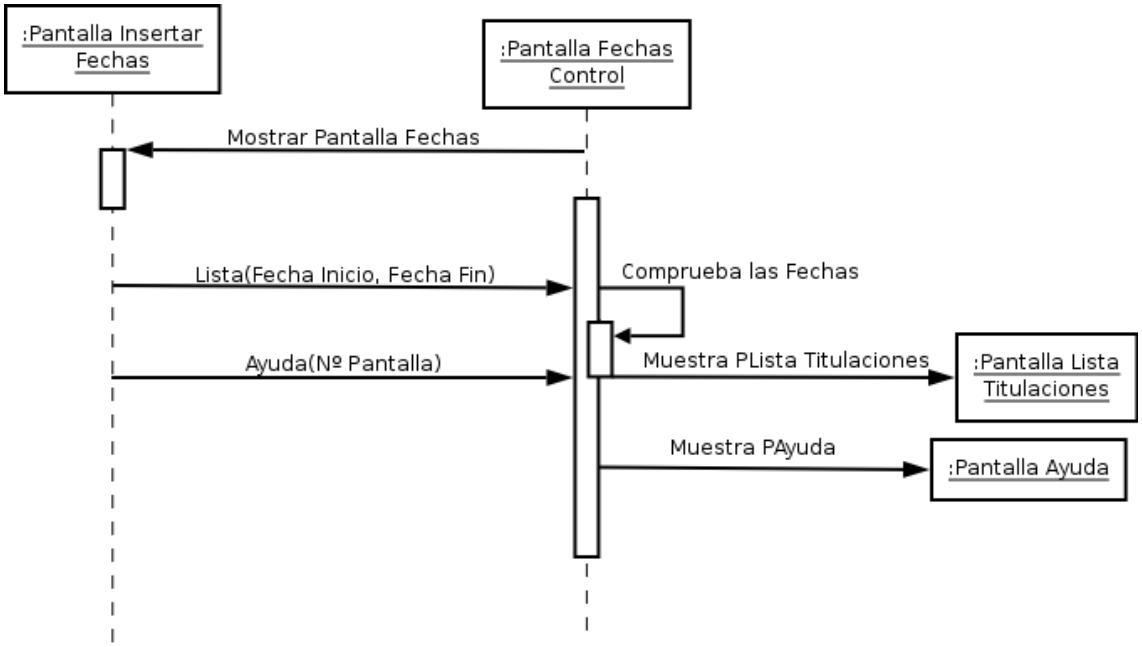

Figura 4.36: Diagrama de Interacción PFechas

El sistema controlará que las fechas del rango del calendario son correctas. Al usuario se le presentan las siguientes operaciones dentro de la pantalla Fechas.

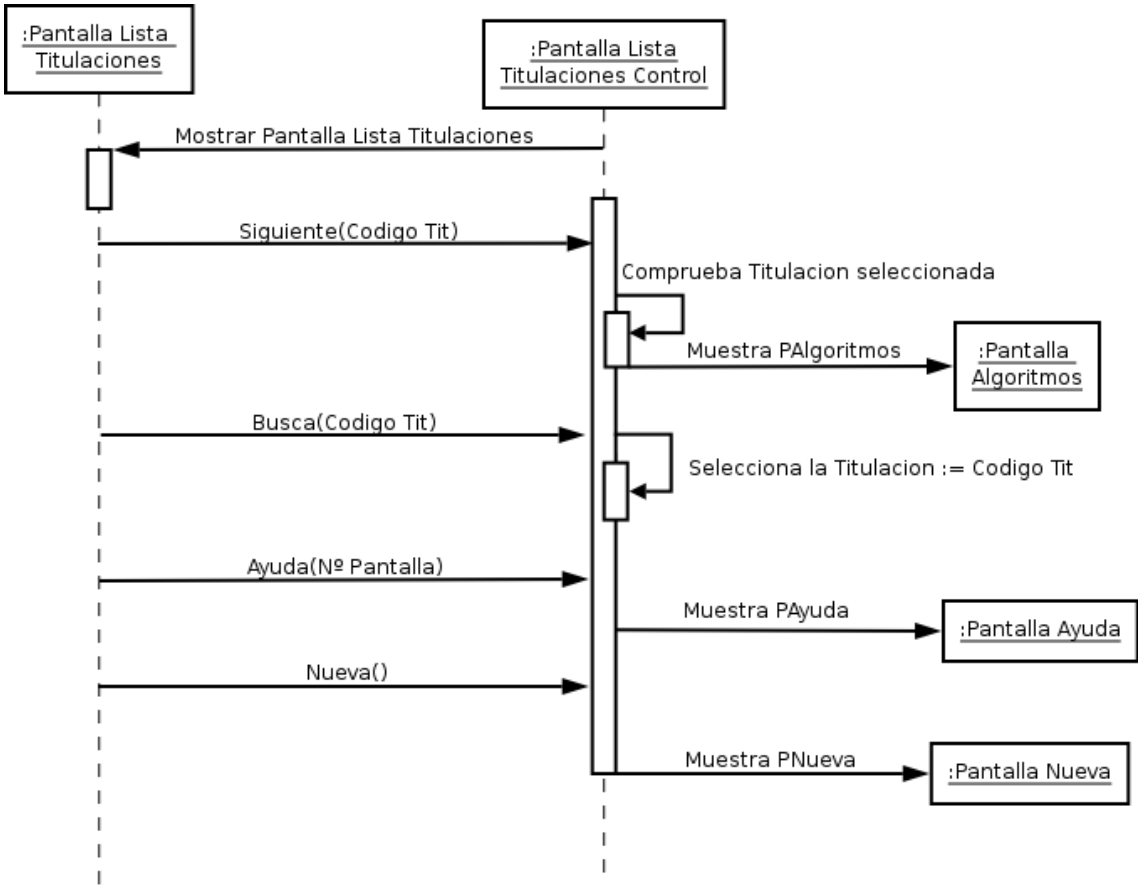

Figura 4.37: Diagrama de Interacción PLista Titulaciones

El sistema controlará que haya una titulación seleccionada antes de pasar a la pantalla Algoritmos. Al usuario se le presentan las siguientes operaciones dentro de la pantalla Lista Titulaciones.

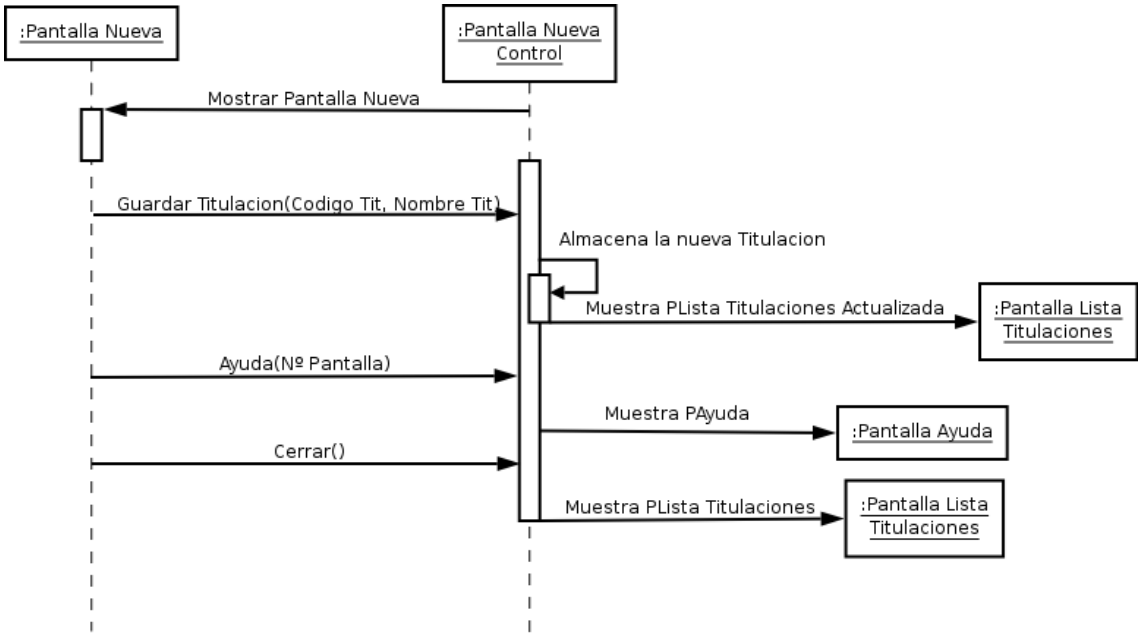

Figura 4.38: Diagrama de Interacción PNueva

El sistema guardará los nuevos datos (Codigo Tit, Nombre Tit) de la nueva Titulación, en el caso de que el usuario decida almacenarla, en caso contrario se cerrará la pantalla PNueva y mostraremos la pantalla Lista Titulaciones sin realizar ninguna modificación.

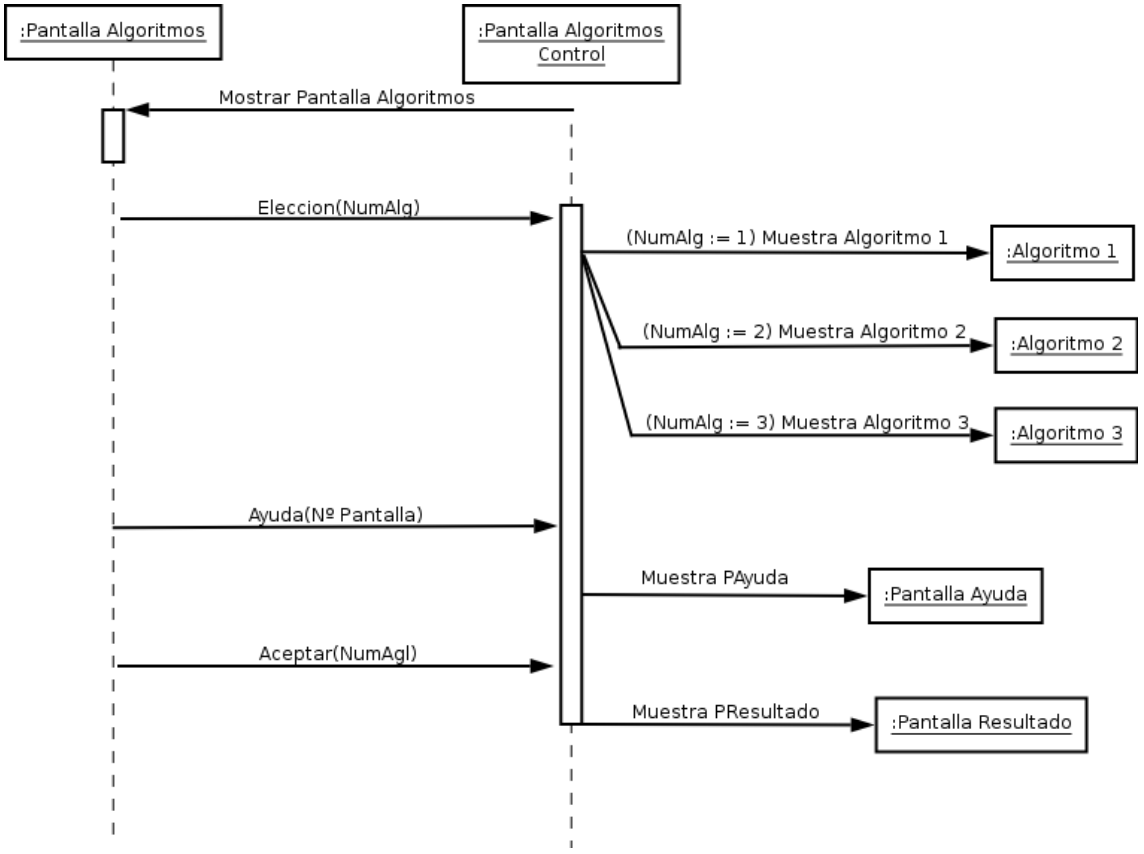

Figura 4.39: Diagrama de Interacción PAlgoritmos

El sistema, según el algoritmo seleccionado llamará a un algoritmo u otro para que realice el calendario con los datos que se le han ido introduciendo con anterioridad a la aplicación. Mostrará en pantalla según la opción (algoritmo seleccionado) la información relativa a cada algoritmo.

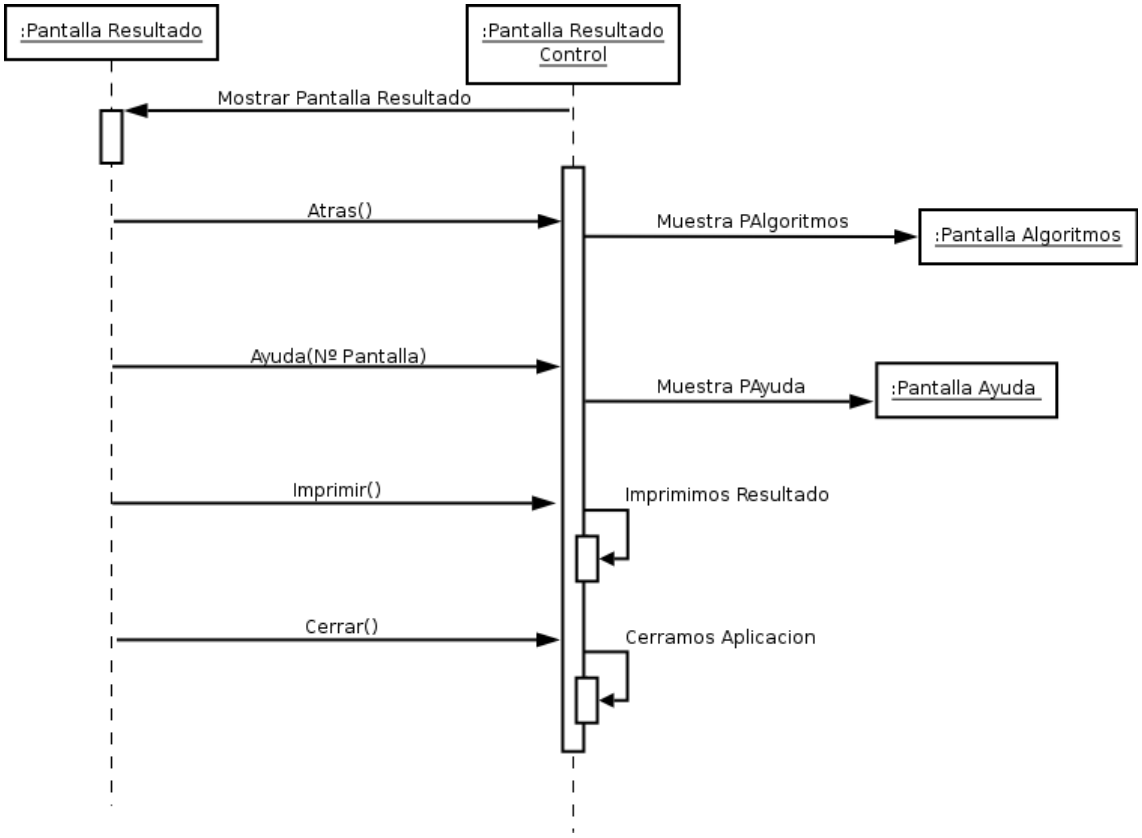

Figura 4.40: Diagrama de Interacción PResultado

El sistema, si lo desea el usuario, volverá a la pantalla PAlgoritmos o imprimirá el resultado que se visualiza en pantalla.

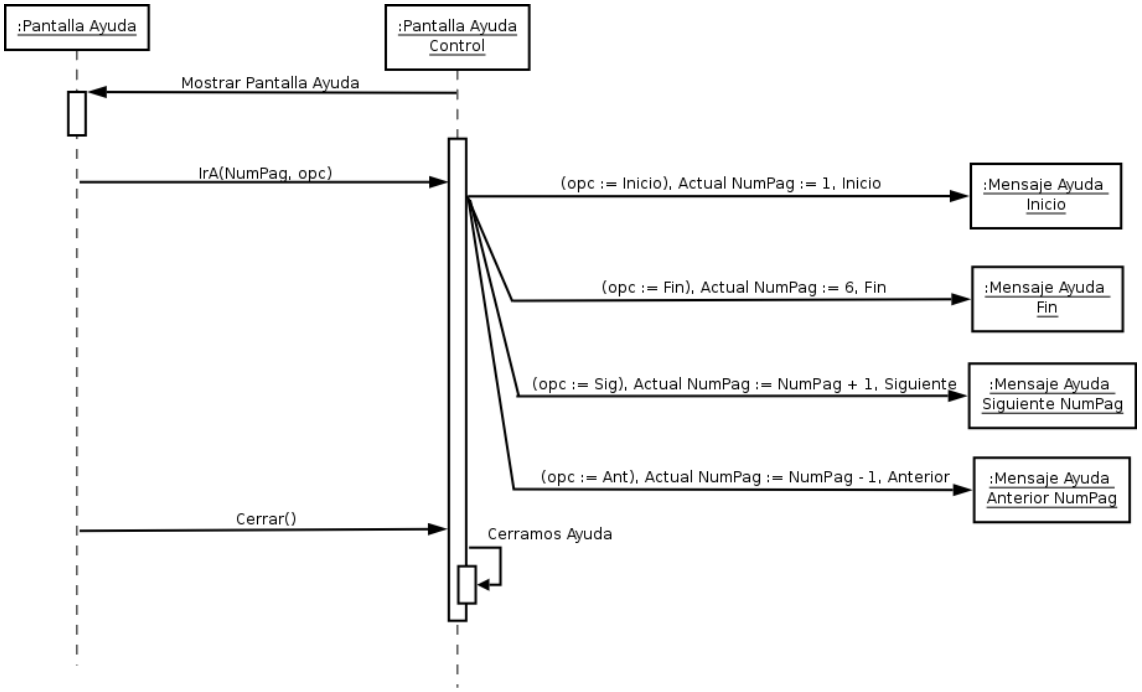

Figura 4.41: Diagrama de Interacción PAyuda

El sistema, controlando la posición en la que nos encontremos en el programa, visualizará la ayuda de esa pantalla, así mismo controlará la posición en la que estemos en la ayuda para navegar por la misma sin afectar al programa principal.

4.3.3. Diseño de la capa de control de la aplicación

4.3.3.1. *Diagramas de secuencia*

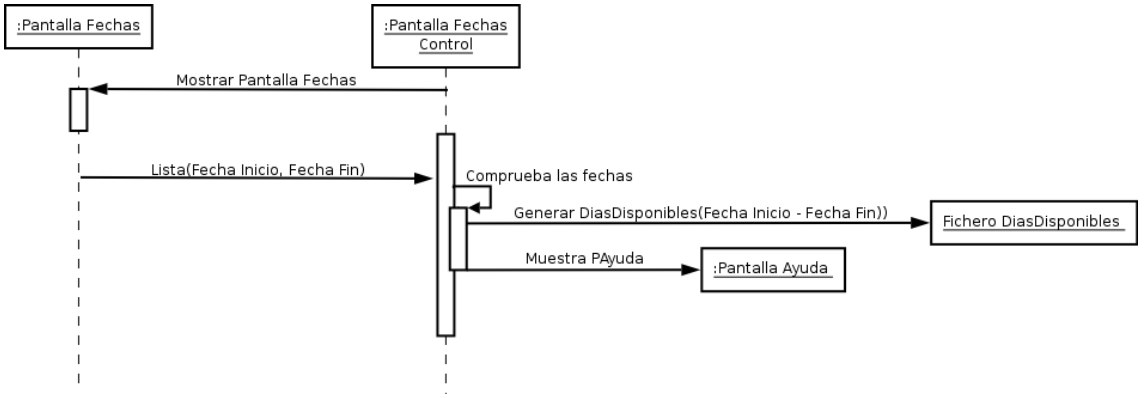

Figura 4.42: Diagrama de Interacción, capa de control en Pantalla Fechas

El sistema almacena, una vez comprobadas las fechas, la diferencia de los días de tal forma que se cuentan multiplicándose por 2, por los 2 turnos de cada día (turno mañana y turno de tarde) y restando 1 turno por el sábado y 2 turnos por el domingo, obteniendo así el número de "huecos" disponibles para generar nuestro calendario.

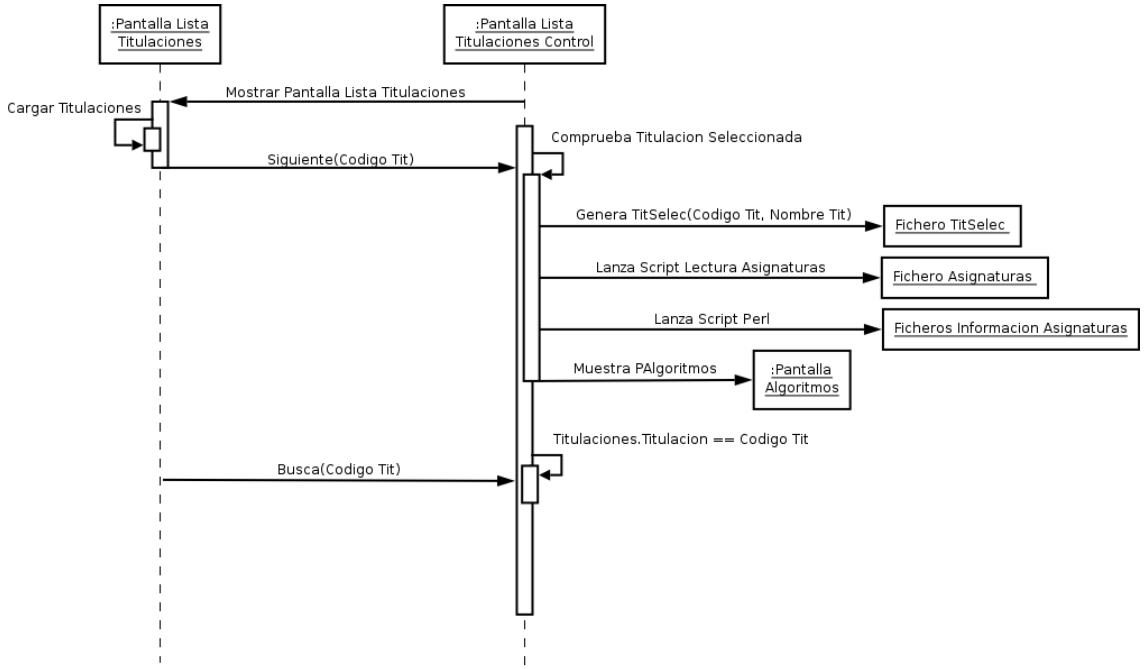

Figura 4.43: Diagrama de Interacción, capa de control en Pantalla Lista Titulaciones

El sistema carga en la lista de Titulaciones el contenido del fichero "Titulaciones" de tal forma que sólo mostramos el nombre de las éstas.

El sistema almacena, una vez comprobada que hay una Titulación seleccionada, el código y nombre de la Titulación.

Teniendo el código de ésta, el sistema lanza un script en el que va leyendo dentro del directorio "Asignaturas" de nuestra aplicación las diferentes asignaturas (anteriormente almacenadas) que empiezan por dicho código (asignaturas de la Titulación). Los nombres y códigos de éstas quedan almacenados en un fichero llamado "AsignaturasTit" que posteriormente utilizaremos.

Finalizado este proceso, lanzaremos un script Perl en el que, con la información del fichero "AsignaturasTit", podremos ir obteniendo la información que nos es necesaria de las asignaturas de la Titulación, almacenándose cada una de ésta en un fichero con el nombre de la Titulación a la que pertenece.

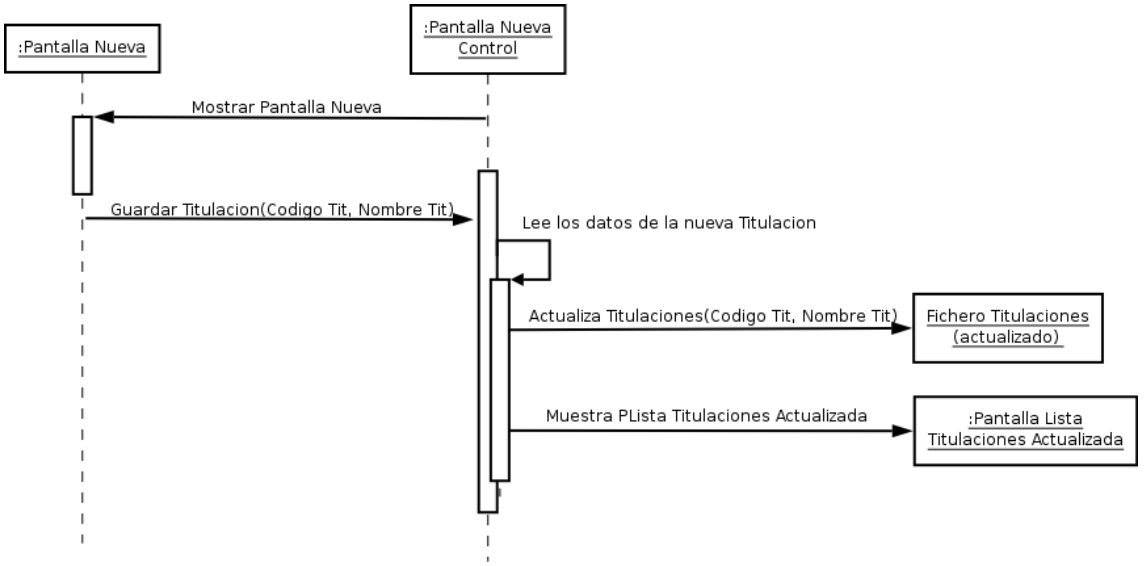

Figura 4.44: Diagrama de Interacción, capa de control en Pantalla Nueva

El sistema almacena, el código y nombre de la nueva Titulación en el fichero "Titulaciones" del programa. Cuando muestra la pantalla "Lista Titulaciones" ésta ya tiene como objetivo cargar el fichero "Titulaciones" con lo que se mostraría la pantalla "Lista Titulaciones" de forma actualizada.

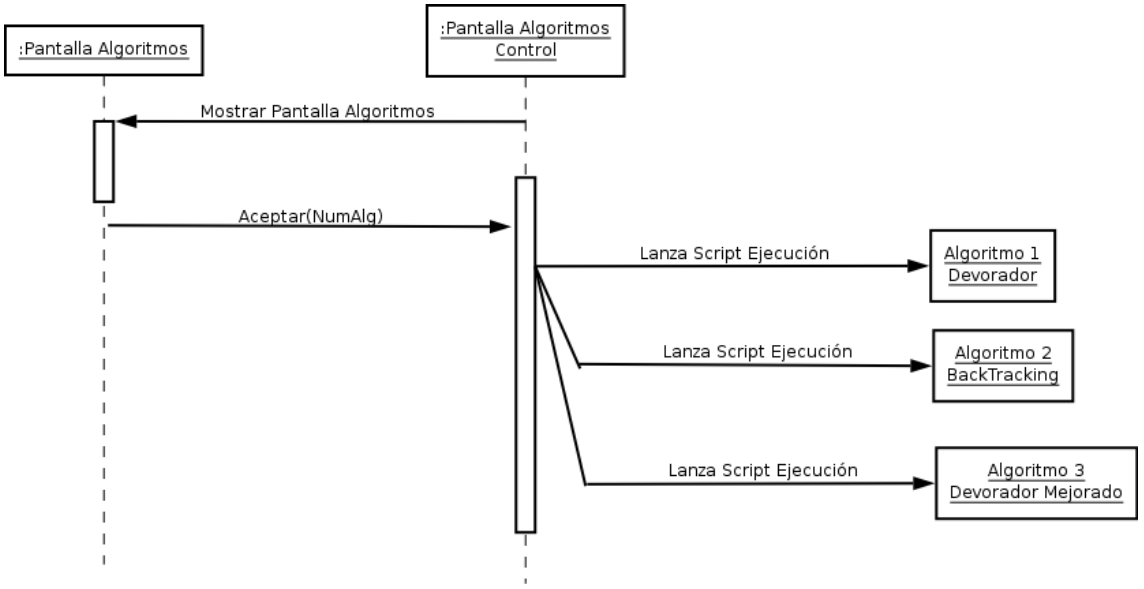

Figura 4.45: Diagrama de Interacción, capa de control en Pantalla Algoritmos

El sistema, según el número del algoritmo que haya sido seleccionado, lanzará un script en el que llamará al algoritmo que ocupe dicha posición para que efectue el coloreo del grafo y así obtener el calendario. (NumAlg = 1, Algoritmo Devorador; NumAlg = 2, Algoritmo BackTrackin; NumAlg = 3, Algoritmo Devorador Mejorado).

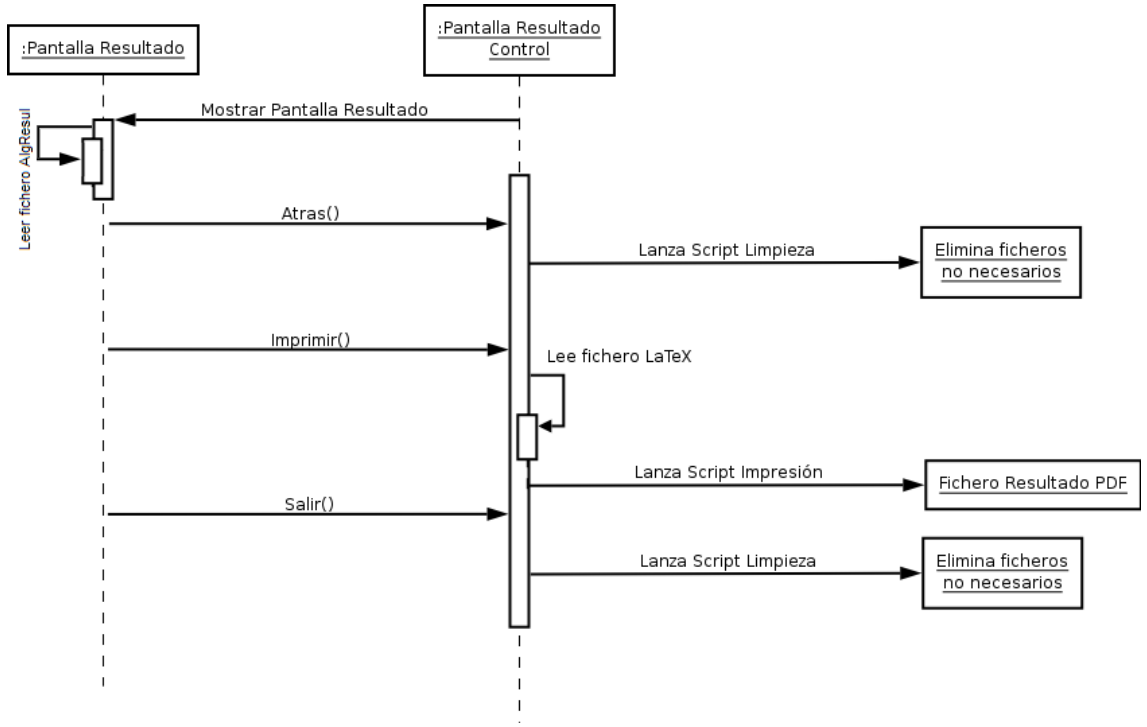

Figura 4.46: Diagrama de Interacción, capa de control en Pantalla Resultado

El sistema, eliminará los ficheros generados por el algoritmo empleado para que el siguiente, en el caso de que se haya pulsado "Atrás", pueda ejecutarse sin ningún problema; del mismo modo que si salimos de la aplicación, el sistema borra estos ficheros para que la próxima vez que ejecutemos nuestra aplicación los resultados sean correctos.

A la hora de imprimir, lee el sistema de un fichero plantilla "Latex" del que tenemos el modelo de nuestra tabla de impresión, generando a partir de este y los datos que aparecen en la pantalla, el fichero PDF resultante.

4.3.4. Diseño de la capa de dominio

4.3.4.1. *Diagramas de secuencia*

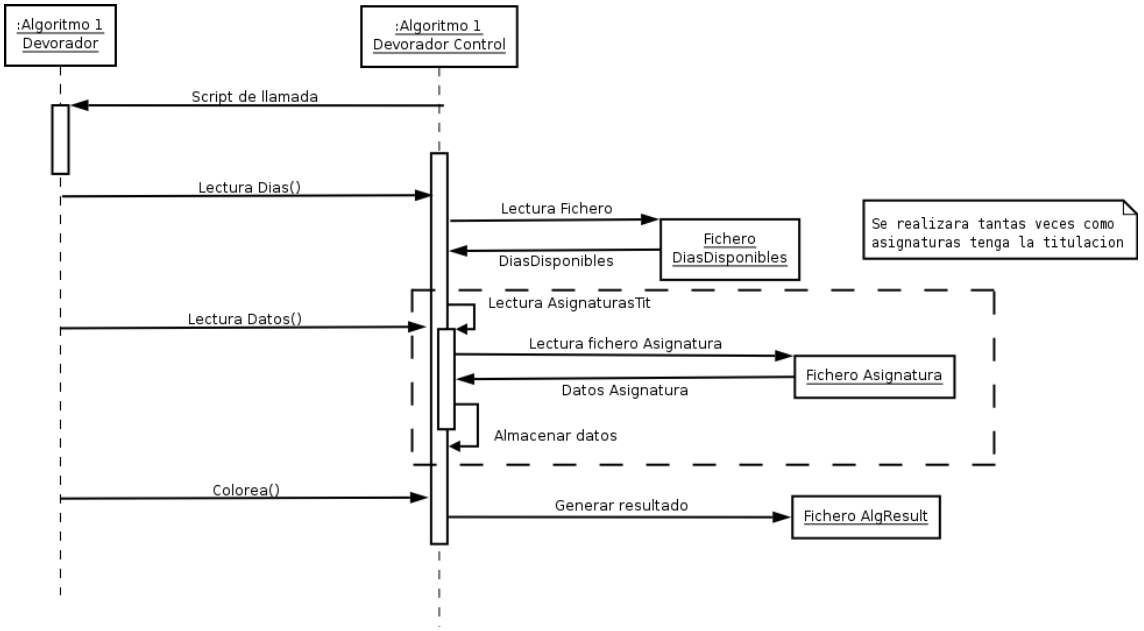

Figura 4.47: Diagrama de Interacción, capa de dominio, Algoritmo 1 Devorador

El sistema, lee los días o huecos disponibles, posteriormente de los ficheros generados antes, con la información de las asignaturas, vamos leyendo los datos que necesitamos (código y alumnos matriculados, su DNI) y se irán almacenando en los nodos del Grafo a colorear que nos estamos creando. Este proceso se realizará hasta que se lean todas las asignaturas de las que dispone la Titulación.

Seguidamente el algoritmo con todos estos datos se dispone a colorear, aquí según funcionamiento del mismo y devuelve el resultado en el fichero "AlgRes" del que posteriormente leeremos la información para mostrarla, bien por pantalla o de forma impresa.

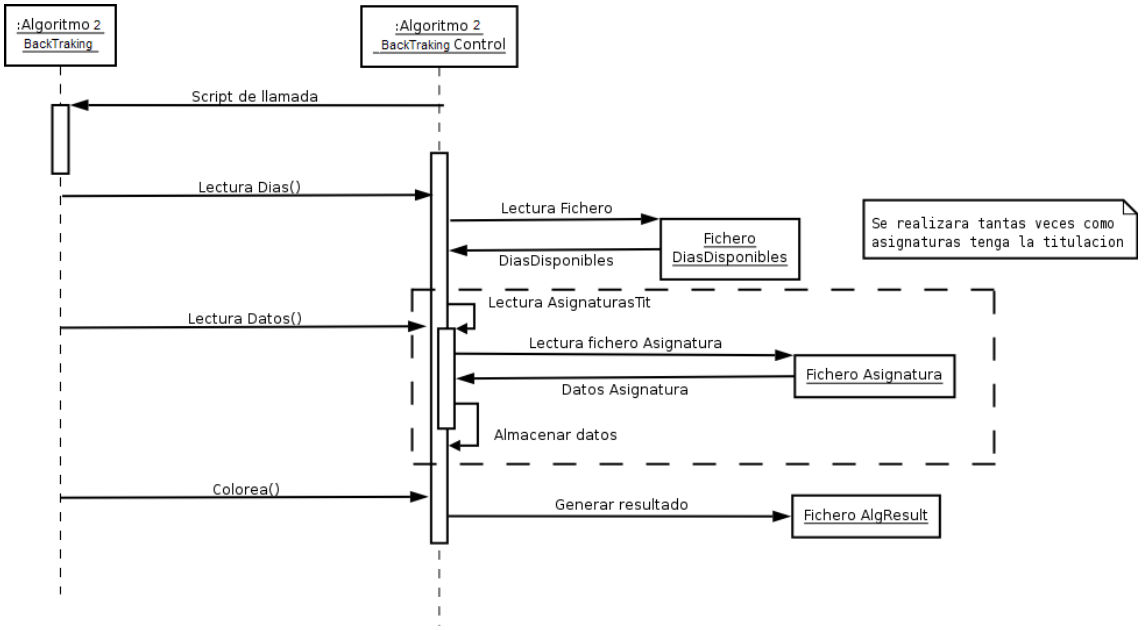

Figura 4.48: Diagrama de Interacción, capa de dominio, Algoritmo 2 BackTracking

El sistema, lee los días o huecos disponibles, posteriormente de los ficheros generados antes, con la información de las asignaturas, vamos leyendo los datos que necesitamos (código y alumnos matriculados, su DNI) y se irán almacenando en los nodos del Grafo a colorear que nos estamos creando. Este proceso se realizará hasta que se lean todas las asignaturas de las que dispone la Titulación.

Seguidamente el algoritmo con todos estos datos se dispone a colorear, aquí según funcionamiento del mismo y devuelve el resultado en el fichero "AlgRes" del que posteriormente leeremos la información para mostrarla, bien por pantalla o de forma impresa.

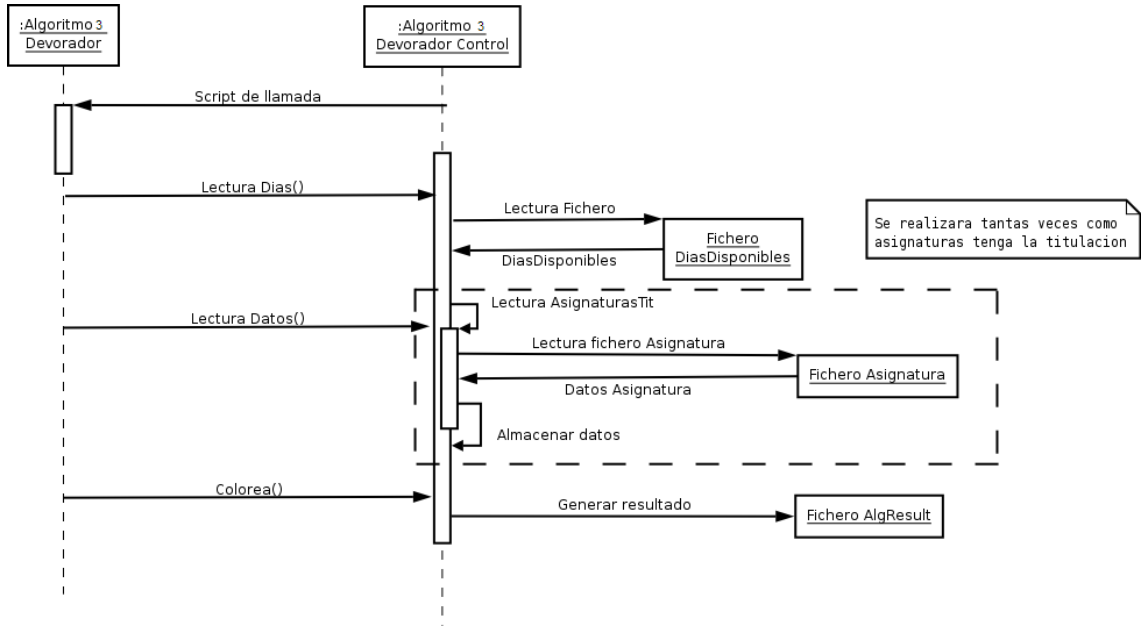

Figura 4.49: Diagrama de Interacción, capa de dominio, Algoritmo 3 Devorador Mejorado

El sistema, lee los días o huecos disponibles, posteriormente de los ficheros generados antes, con la información de las asignaturas, vamos leyendo los datos que necesitamos (código y alumnos matriculados, su DNI) y se irán almacenando en los nodos del Grafo a colorear que nos estamos creando. Este proceso se realizará hasta que se lean todas las asignaturas de las que dispone la Titulación.

Seguidamente el algoritmo con todos estos datos se dispone a colorear, aquí según funcionamiento del mismo y devuelve el resultado en el fichero "AlgRes" del que posteriormente leeremos la información para mostrarla, bien por pantalla o de forma impresa.

4.4. Pruebas y validación

Las pruebas a realizar con los algoritmos es comprobar su tiempo de ejecución en un número determinado de veces para comprobar de forma más o menos exacta su rendimiento respecto al tiempo.

4.4.1. Primera prueba

En nuestra primera prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 10 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,008

Algoritmo BackTracking: 0,03

Algoritmo Devorador Mejorado: 0,006

A pesar de las pocas iteraciones realizadas en esta prueba, podemos observar que se presenta una leve diferencia entre los algoritmos Devoradores y el algoritmo BackTracking.

4.4.2. Segunda prueba

En nuestra segunda prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 50 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,008

Algoritmo BackTracking: 0,131

Algoritmo Devorador Mejorado: 0,008

Como podemos observar, al quintuplar el número de iteraciones la distancia entre algoritmos queda más reflejada, aunque todavía es poca la diferencia entre estos. El algoritmo Devorador Mejorado no presenta diferencia con el primero, aunque más adelante veremos un atisbo de retardo, al constar de pocos nodos, no se verá muy bien reflejado en este estudio la diferencia entre estos algoritmos, se podría decir que casi utilizaríamos el mismo.

4.4.3. Tercera prueba

En nuestra tercera prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 100 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,014

Algoritmo BackTracking: 0,337

Algoritmo Devorador Mejorado: 0,012

4.4.4. Cuarta prueba

En nuestra cuarta prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 200 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,018

Algoritmo BackTracking: 0,545

Algoritmo Devorador Mejorado: 0,022

En esta prueba es donde se ha reflejado un poco el coste en tiempo del algoritmo Devorador Mejorado, aunque repetimos que ésta es muy poco significativa ya que disponemos de pocas asignaturas para poder hacer las pruebas de forma correcta.

4.4.5. Quinta prueba

En nuestra quinta prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 100 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,042

Algoritmo BackTracking: 1,413

Algoritmo Devorador Mejorado: 0,045

4.4.6. Sexta prueba

En nuestra sexta prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 1000 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,075

Algoritmo BackTracking: 2,795

Algoritmo Devorador Mejorado: 0,071

Como observamos entre los dos algoritmos Deboradores vuelven a salir tiempos muy semejantes, siendo incluso el del Devorador Mejorado menor. El algoritmo BackTracking llega ya a una media de 2.795 segundos, mientras que los Deboradores no han llegado a las décimas de segundos.

4.4.7. Séptima prueba

En nuestra séptima prueba con un conjunto de datos de 21 asignaturas, 21 nodos del grafo a colorear y con un número indeterminado de alumnos matriculados, repetimos la operación 2000 veces. Aquí los resultados después de aplicar la media:

Algoritmo Devorador: 0,158

Algoritmo BackTracking: 5,043

Algoritmo Devorador Mejorado: 0,158

Como observamos entre los dos algoritmos Deboradores las diferencias son casi inexistentes por lo ya comentado. Comparándolos con el algoritmo BackTracking, han tenido que suceder 2000 iteraciones para que llegen a las décimas de segundo, mientras que con el BackTracking nos situamos con un coste en el tiempo de más de 5 segundos.

Obviamente se ve la mejora respecto al tiempo entre los algoritmos Devoradores y el de BackTracking. No podemos decir lo mismo de la comparación entre los dos Devoradores.

4.5. Gráficos de las pruebas

Aquí mostramos los gráficos de las pruebas obtenidas después de realizar el estudio sobre el coste del tiempo de cada uno de los algoritmos. De esta forma se puede ver de forma más clara la evolución de las iteraciones en los diferentes algoritmos.

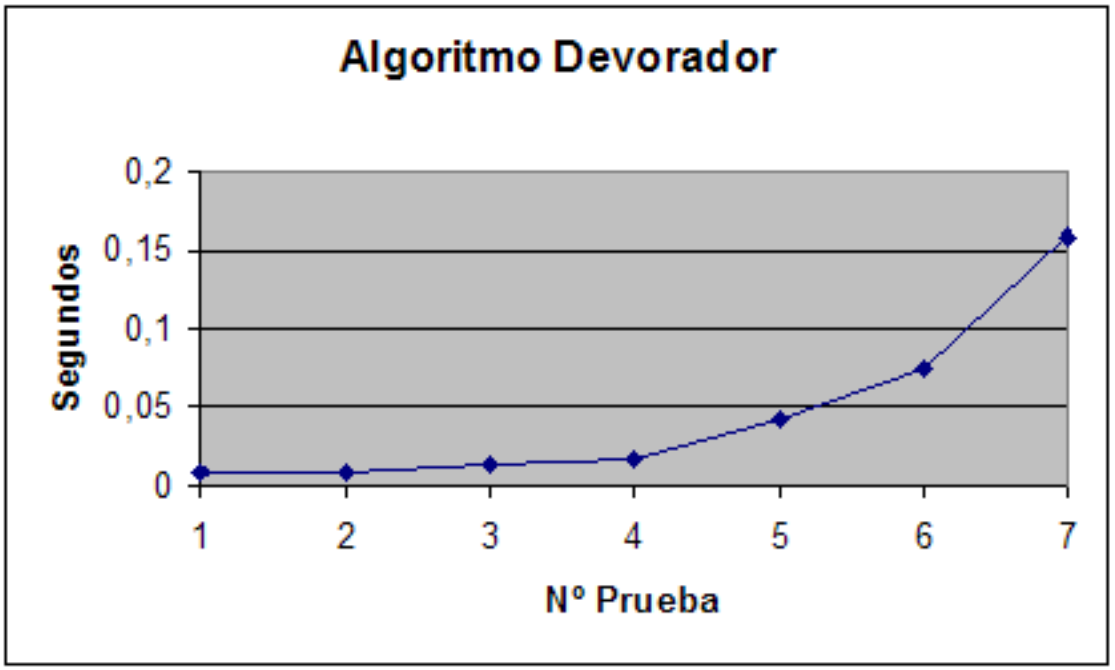

Figura 4.50: Estudio del tiempo en el algoritmo Devorador

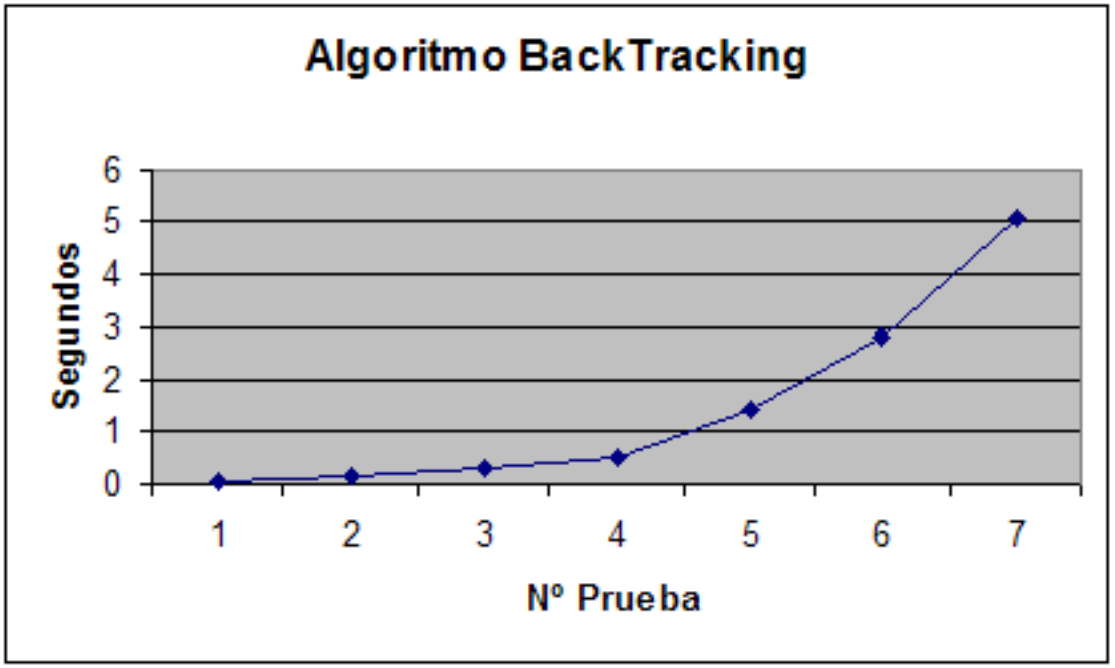

Figura 4.51: Estudio del tiempo en el algoritmo BackTracking

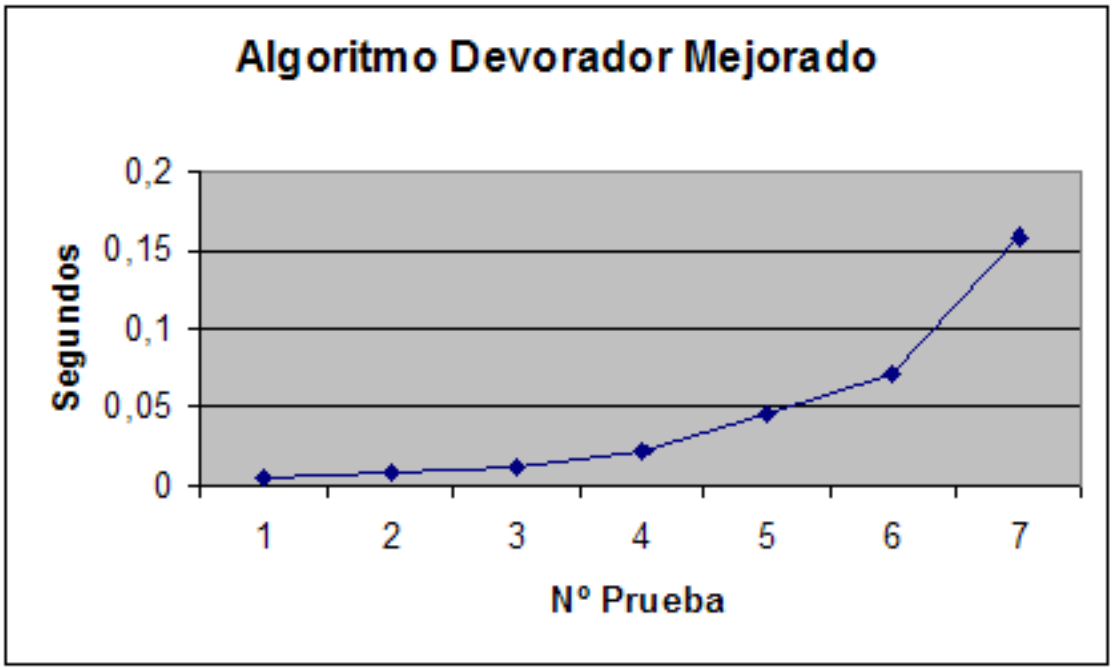

Figura 4.52: Estudio del tiempo en el algoritmo Devorador Mejorado

4.6. Documentación de la ejecución de las pruebas

Aquí adjuntamos los tiempos obtenidos en cada una de las pruebas con los diferentes ciclos para observar mejor los resultados:

4.6.1. Datos Prueba1: 10 iteraciones

4.6.1.1. Algoritmo Devorador

Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0
Tiempo : 0
Tiempo : 0.01
Tiempo : 0
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01

4.6.1.2. Algoritmo BackTracking

Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.03
Tiempo : 0.05
Tiempo : 0.03
Tiempo : 0.02
Tiempo : 0.03

4.6.1.3. Algoritmo Devorador Mejorado

Tiempo : 0.01
Tiempo : 0
Tiempo : 0
Tiempo : 0
Tiempo : 0
Tiempo : 0.01
Tiempo : 0.01

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01

4.6.2. Datos Prueba2: 50 iteraciones

4.6.2.1. Algoritmo Devorador

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0
Tiempo : 0.01
Tiempo : 0
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01

4.6.2.2. Algoritmo BackTrackin

Tiempo : 0.13
Tiempo : 0.14
Tiempo : 0.14
Tiempo : 0.14
Tiempo : 0.13
Tiempo : 0.13
Tiempo : 0.13
Tiempo : 0.13
Tiempo : 0.12
Tiempo : 0.12

4.6.2.3. Algoritmo Devorador Mejorado

Tiempo : 0
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01

Tiempo : 0.01

4.6.3. Datos Prueba3: 100 iteraciones

4.6.3.1. Algoritmo Devorador

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01

4.6.3.2. Algoritmo BackTracking

Tiempo : 0.25
Tiempo : 0.28
Tiempo : 0.28
Tiempo : 0.3
Tiempo : 0.31
Tiempo : 0.25
Tiempo : 0.27
Tiempo : 0.28
Tiempo : 0.28
Tiempo : 0.29
Tiempo : 0.27
Tiempo : 0.31

4.6.3.3. Algoritmo Devorador Mejorado

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.01

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.01

4.6.4. Datos Prueba4: 200 iteraciones

4.6.4.1. Algoritmo Devorador

Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.04
Tiempo : 0.01
Tiempo : 0.02

4.6.4.2. Algoritmo BackTracking

Tiempo : 0.57
Tiempo : 0.54
Tiempo : 0.54
Tiempo : 0.54
Tiempo : 0.55
Tiempo : 0.57
Tiempo : 0.54
Tiempo : 0.52
Tiempo : 0.52
Tiempo : 0.56

4.6.4.3. Algoritmo Devorador Mejorado

Tiempo : 0.03
Tiempo : 0.02
Tiempo : 0.03
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.03
Tiempo : 0.02

Tiempo : 0.03

4.6.5. Datos Prueba5: 500 iteraciones

4.6.5.1. Algoritmo Devorador

Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.05
Tiempo : 0.04
Tiempo : 0.05
Tiempo : 0.04

4.6.5.2. Algoritmo BackTracking

Tiempo : 1.42
Tiempo : 1.41
Tiempo : 1.43
Tiempo : 1.42
Tiempo : 1.41
Tiempo : 1.45
Tiempo : 1.39
Tiempo : 1.4
Tiempo : 1.4
Tiempo : 1.4

4.6.5.3. Algoritmo Devorador Mejorado

Tiempo : 0.05
Tiempo : 0.04
Tiempo : 0.09
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.03
Tiempo : 0.04
Tiempo : 0.04
Tiempo : 0.04

4.6.6. Datos Prueba6: 1000 iteraciones

4.6.6.1. Algoritmo Devorador

Tiempo : 0.06
Tiempo : 0.07
Tiempo : 0.07
Tiempo : 0.08
Tiempo : 0.08
Tiempo : 0.07
Tiempo : 0.08
Tiempo : 0.08
Tiempo : 0.08
Tiempo : 0.08

4.6.6.2. Algoritmo BackTracking

Tiempo : 2.84
Tiempo : 2.79
Tiempo : 2.81
Tiempo : 2.82
Tiempo : 2.81
Tiempo : 2.82
Tiempo : 2.77
Tiempo : 2.78
Tiempo : 2.74
Tiempo : 2.77

4.6.6.3. Algoritmo Devorador Mejorado

Tiempo : 0.08
Tiempo : 0.09
Tiempo : 0.07
Tiempo : 0.07
Tiempo : 0.07
Tiempo : 0.06
Tiempo : 0.06
Tiempo : 0.07
Tiempo : 0.07
Tiempo : 0.07

4.6.7. Datos Prueba7: 2000 iteraciones

4.6.7.1. Algoritmo Devorador

Tiempo : 0.16
Tiempo : 0.17
Tiempo : 0.16
Tiempo : 0.17
Tiempo : 0.15
Tiempo : 0.15
Tiempo : 0.15
Tiempo : 0.15
Tiempo : 0.16
Tiempo : 0.16

4.6.7.2. Algoritmo BackTracking

Tiempo : 5.69
Tiempo : 5.58
Tiempo : 5.58
Tiempo : 5.55
Tiempo : 5.53
Tiempo : 5.66
Tiempo : 5.68
Tiempo : 5.63
Tiempo : 5.6
Tiempo : 5.61

4.6.7.3. Algoritmo Devorador Mejorado

Tiempo : 0.15
Tiempo : 0.16
Tiempo : 0.15
Tiempo : 0.21
Tiempo : 0.15
Tiempo : 0.15
Tiempo : 0.15
Tiempo : 0.15
Tiempo : 0.16
Tiempo : 0.15

4.7. Pruebas adicionales

4.7.1. 150 Iteraciones

4.7.1.1. Algoritmo Devorador

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.03
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02

4.7.1.2. Algoritmo BackTracking

Tiempo : 0.41
Tiempo : 0.39
Tiempo : 0.39
Tiempo : 0.38
Tiempo : 0.44
Tiempo : 0.42
Tiempo : 0.42
Tiempo : 0.43
Tiempo : 0.43
Tiempo : 0.42

4.7.1.3. Algoritmo Devorador Mejorado

Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.02
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01
Tiempo : 0.01

4.7.2. 300 Iteraciones

4.7.2.1. Algoritmo Devorador

Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.02
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.03

4.7.2.2. Algoritmo BackTracking

Tiempo : 0.84
Tiempo : 0.82
Tiempo : 0.82
Tiempo : 0.9
Tiempo : 0.83
Tiempo : 0.84
Tiempo : 0.83
Tiempo : 0.85
Tiempo : 0.85
Tiempo : 0.82

4.7.2.3. Algoritmo Devorador Mejorado

Tiempo : 0.02
Tiempo : 0.02
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.02
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.03
Tiempo : 0.03

Capítulo 5

Resumen

Gracias a la creación de este proyecto, se ha podido profundizar más en el estudio de los lenguajes con los que hemos realizado esta aplicación a la vez que hemos conocido otros nuevos.

La creación de este programa empezó con la implementación de lo que iba a ser la base de nuestra aplicación, los grafos y posteriormente los algoritmos con los que íbamos a trabajar, emplearíamos c++. Obviamente antes estudiamos cada uno de estos viendo las características de cada uno.

Una vez finalizados, nos planteamos las opciones que debería de tener nuestro usuario para poder generar los calendarios; se nos presenta el problema de la Base de Datos de los alumnos. Lo solventamos con el acceso a la página web de la UCA "red-campus", aunque en futuras versiones queremos que sea posible acceder a la base de datos. Para obtener los datos de la web empleamos el lenguaje Perl, con el que ya nos generamos nuestros propios ficheros con la información que nos conscierne.

De forma paralela al programa (mediante script) es como manejamos toda esta información, de ahí a que tengamos que esperar algunos segundos para que los datos se procesen.

Finalmente y según el algoritmo seleccionado por el usuario, llamaremos a aquél que será el responsable de darnos una solución, mostrándola por pantalla en nuestra aplicación.

Todos estos pasos llevan un estudio previo sobre las diferentes opciones que podíamos abarcar para que nuestro programa fuese más eficiente e intuitivo para el usuario.

Se ha conseguido cumplir todos los objetivos que nos marcamos, el evitar segundas vueltas y que en caso de haberlas afectasen al menor número de alumnos matriculados en la titulación. Con esto evitamos elaboración de exámenes del mismo nivel de dificulad, que los alumnos cambien su calendario de exámenes...

Capítulo 6

Conclusiones

Este programa ha sido creado para generar de una forma más rápida y eficiente los calendarios de exámenes de cualquier titulación. La aplicación "Calendarios" intenta evitar lo máximo posible las segundas vueltas y en el caso de haberlas afectará al menor número posible de alumnos. El programa nos ofrece tres algoritmos que según las necesidades del usuario podrá emplear según le convenga.

Como ya comentamos antes, el mayor inconveniente de este programa es el acceso a la base de datos de los alumnos (cuya información es confidencial) y no tenemos otra forma que acceder a estos datos mediante la página web de la UCA "red-campus" y dependemos del acceso a esta. Esto sería una mejora en una futura aplicación a la vez que el análisis estadístico que comentamos de años anteriores.

"Calendarios" nos ofrece un entorno agradable y bastante intuitivo que con la ayuda del programa y su manual se nos presenta facil de manejar. Se nos ofrece la opción de que, una vez introducidas las asignaturas de la titulación, podemos generar tantos calendarios de exámenes para todas las fechas que necesitemos, basándose en los diferentes algoritmos que se nos ofrecen.

Capítulo 7

Manual de Usuario

7.1. Calendarios paso a paso

Bienvenido al manual del programa "Calendarios", en él le describiremos poco a poco todos y cada uno de los pasos que tiene que realizar para la perfecta ejecución de la aplicación.

7.1.1. Antes de ejecuar la aplicación

Debido a que no tenemos acceso a la Base de Datos de los alumnos, tenemos que ingeniárnola para poder obtener la información que necesitamos, la cual es conocer todos y cada uno de los alumnos matriculados en las asignaturas de la titulación de la que vayamos a realizar el calendario. ¿Cómo lo conseguimos? Daremos los siguientes pasos:

a) Abriremos el navegador con el que estemos más familiarizado (no podremos usar mozilla ya que con dicho navegador no se visualiza correctamente la página que vamos a visitar).

b) Introduciremos la dirección "http:redcampus.uca.es". Con esta página ya estamos familiarizados, y como profesor o administrador, podremos tener acceso a la información que buscamos. Registrándonos como tales entraremos en el apartado de "lista de alumnos" y podremos visualizar todas las asignaturas de la titulación (si somos un profesor sólo visualizaremos las de nuestro departamento).

c) Seleccionamos la asignatura.

d) Nos situamos sobre la tabla donde aparecen todos los alumnos matriculados.

e) Pulsaremos con el botón derecho del ratón de tal forma que se nos aparecerá un menú emergente del que seleccionaremos la opción de "Guardar página".

f) Esta página tendremos que almacenarla dentro de la carpeta "Asignaturas" de nuestro programa y cuyo nombre será el código de la misma, es decir

si tenemos la asignatura de Análisis y Diseño de Algoritmos I cuyo código es: 1710008 (hablamos de la titulación de Ingeniería Técnica Informática de Gestión), la almacenaremos en el anterior directorio con el nombre de "1710008.html".

Repetiremos los pasos (del tres al seis) con todas y cada una de las asignaturas de la titulación que queremos realizar el calendario.

Recordamos que todo esto se tiene que llevar a cabo debido a que no tenemos acceso a la Base de Datos de los alumnos ya que esta información es confidencial. Debemos actualizar cada año estos datos ya que los alumnos matriculados cambiarán cada año, durante este año no tendremos porqué modificar estos ficheros ya que son los mismos alumnos del curso y podemos almacenarlas y hacer pruebas de tantos horarios como queramos durante distintas etapas de todo el curso.

7.1.2. ¡Calendarios!

Una vez que tenemos todas las asignaturas almacenadas en nuestro directorio damos paso a ejecutar nuestro programa. La primera pantalla que nos aparece es la siguiente:

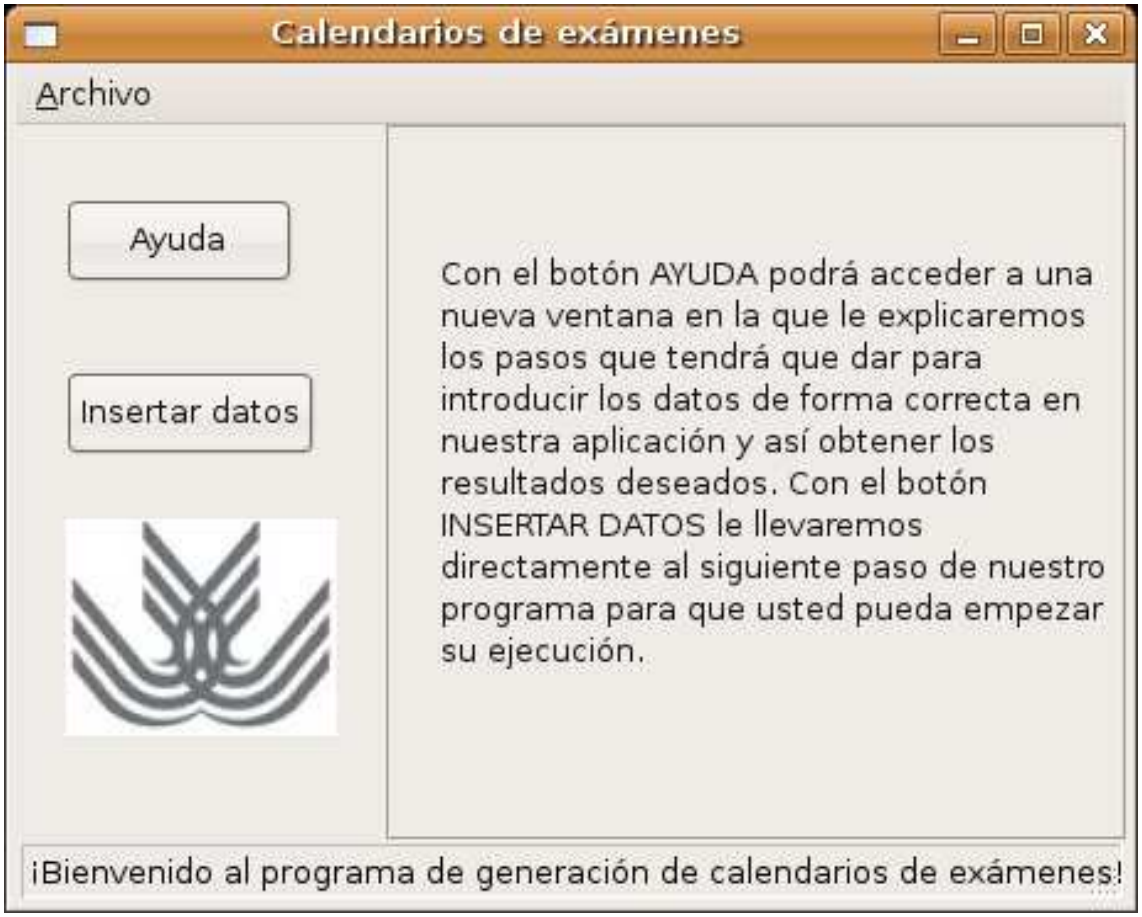

Figura 7.1: Pantalla de Inicio

En ella podemos hacer lo siguiente; podemos solicitar información sobre el programa y se nos aparecerá una breve descripción del objetivo del programa "Calendarios" lo conseguiremos pulsando en "Archivo/Sobre...". Nos aparece sobre la pantalla inicial el siguiente mensaje:

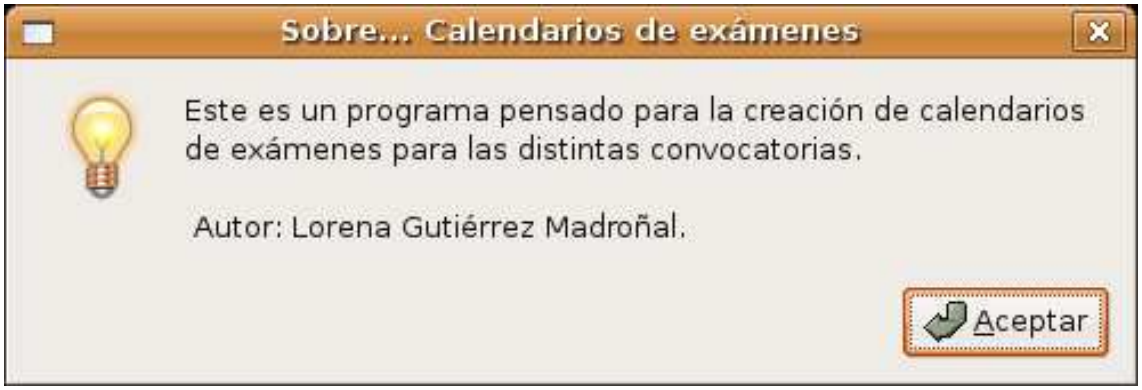

Figura 7.2: Pantalla de Información

Si volvemos a la pantalla de inicio, podremos solicitar ayuda mediante el botón "Ayuda" que aparece en pantalla o bien en el menú de "Archivo". Esta ayuda está disponible a cada paso de la aplicación, por si en algún paso de la misma no estamos seguro de lo que debemos hacer. En ella podemos navegar según nuestras necesidades, aunque según la pantalla en la que nos situemos, al solicitarla, esta mostrará la información correspondiente a dicha parte del programa. Lo veremos en el apartado de Ayuda de forma más detallada.

Y por último podremos pasar al siguiente paso de la aplicación, en el que empezaremos a introducir los datos necesarios para realizar el calendario de exámenes.

7.1.3. ¡Fechas, fechas, fechas!

Pasemos a la pantalla de "Inserción de datos" de la aplicación. En ella se nos indica que tenemos que introducir el rango del calendario de exámenes. Hay dos formas de hacerlo: escribiendo el día, mes y año, según el formato que se nos aparece en la pantalla (nos muestra el día actual) o bien desplegando la pestaña correspondiente y escogiendo nosotros mismos el día de comienzo y fin del calendario.

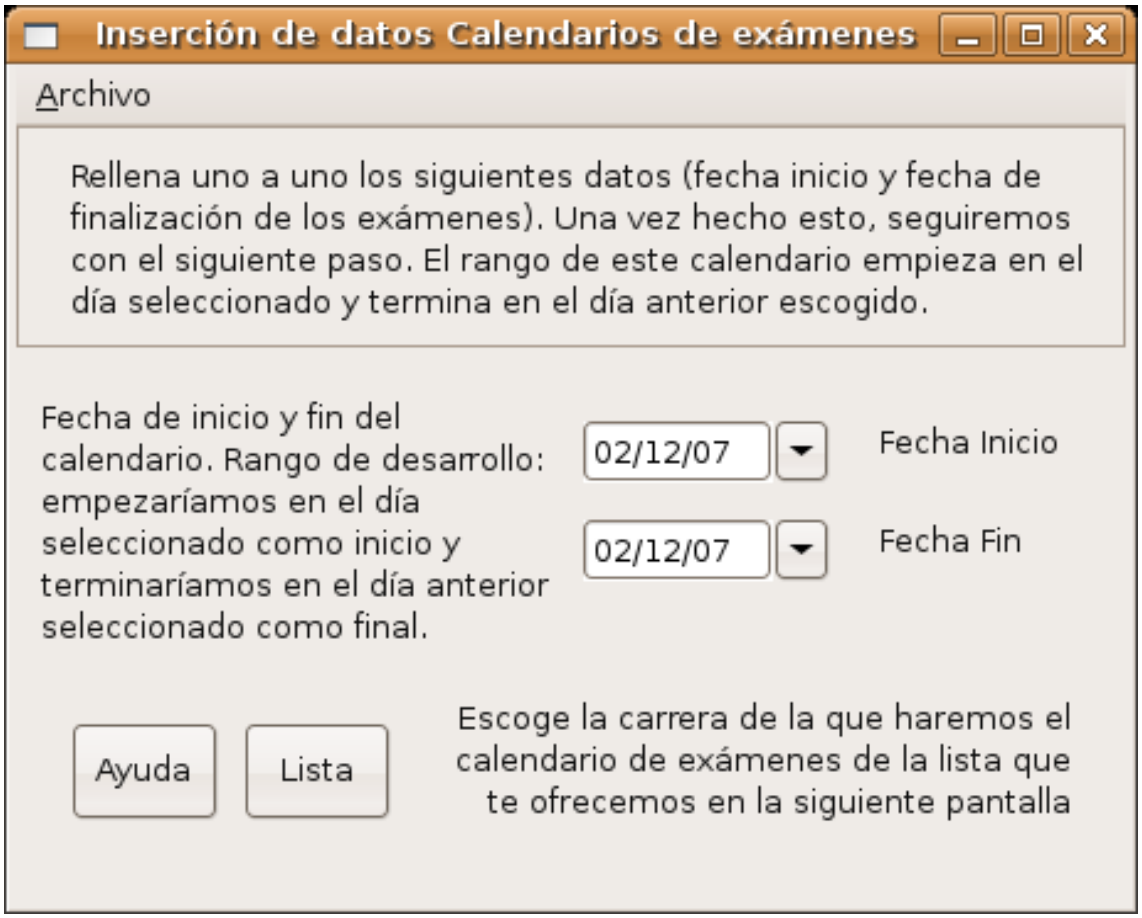

Figura 7.3: Pantalla de Introducción de datos

Del mismo modo que en la pantalla anterior, y como ya hemos indicado, podremos solicitar la ayuda del programa bien pulsando dentro del menú "Archivo" o bien en el botón de "Ayuda" que aparece en pantalla. De esta forma se nos aparecerá la ayuda referente a esta página si no sabemos en un momento dado el paso que tenemos quedar.

Pasaremos al siguiente paso pulsando el botón “Lista”, antes de pasar a la siguiente pantalla, el sistema comprobará si las fechas introducidas son correctas; en caso afirmativo pasaremos a la pantalla “Lista de Titulaciones” y en caso contrario nos mostrará el siguiente mensaje:

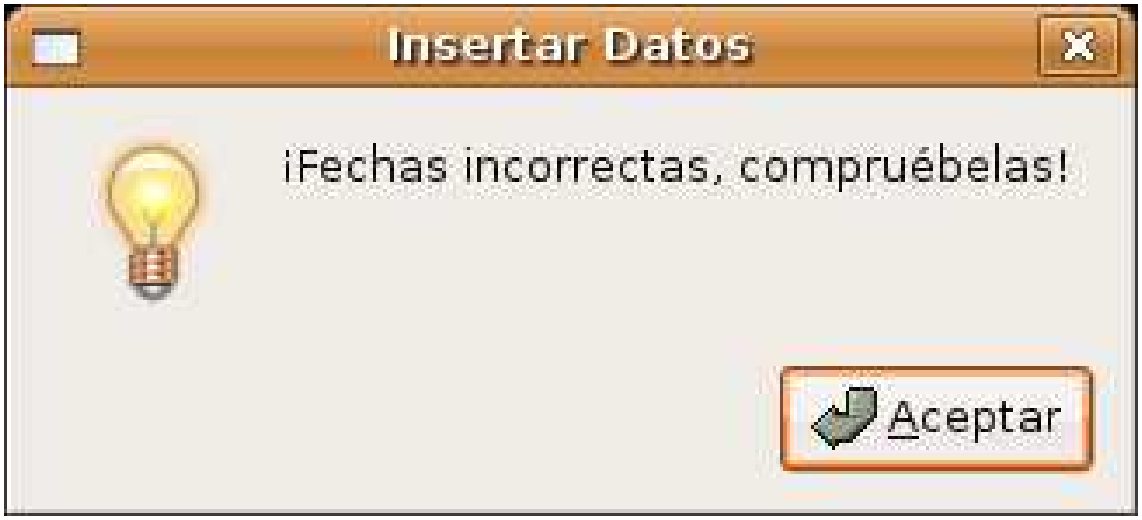

Figura 7.4: Pantalla Fechas erróneas

7.1.4. Escoge la titulación

Ahora se nos muestran una lista de Titulaciones, nuestro siguiente paso será seleccionar una de las que se nos ofrecen, o en caso de que la titulación no se encuentre en la lista, la introduciremos en nuestro programa.

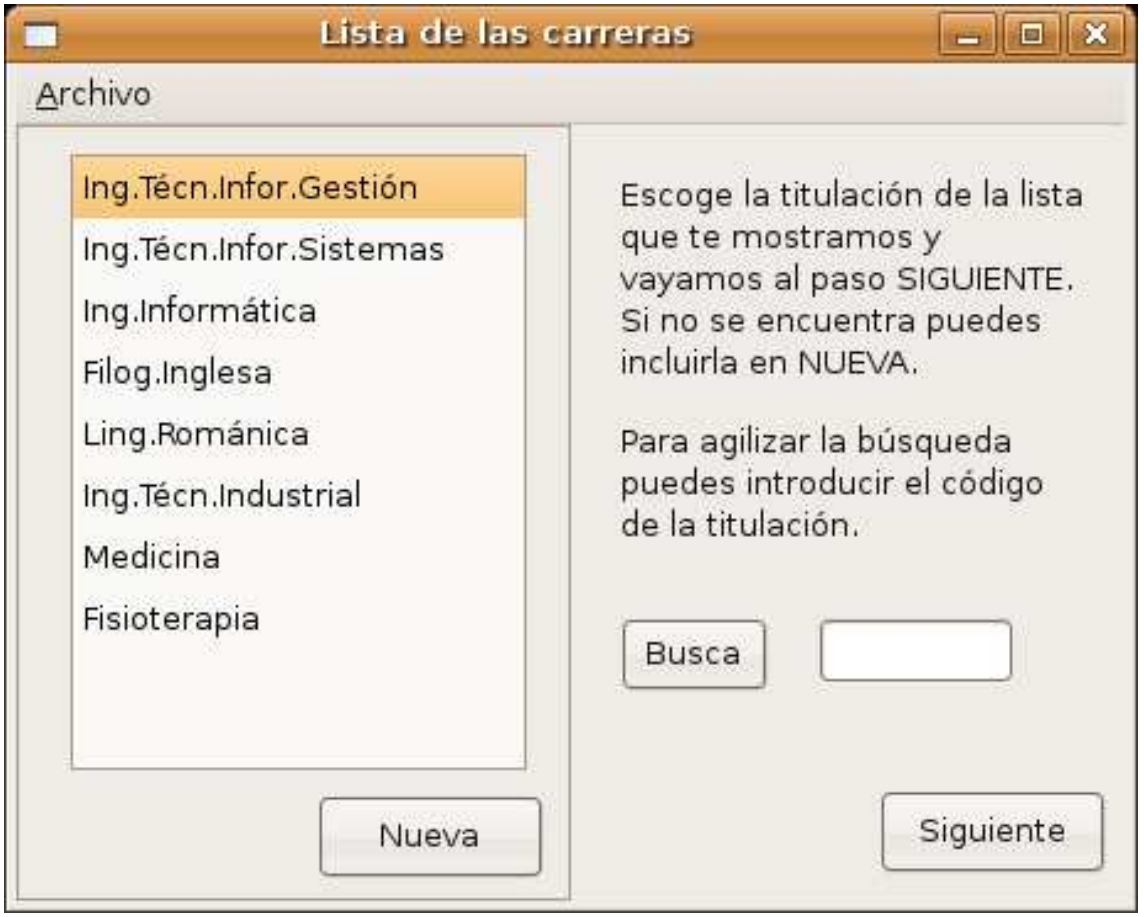

Figura 7.5: Pantalla Lista de Titulaciones

Si a simple vista no encontramos la titulación que queremos, podemos buscarla mediante el código de la misma (obviamente tendremos que saber el código de esta, si no sería imposible buscarla). Introduciremos en el casillero el código de la titulación, en caso que ésta se encuentre se nos mostrará seleccionada en la lista, en caso contrario nos aparecerá el siguiente mensaje invitándonos a introducir la nueva titulación en nuestra aplicación:

Figura 7.6: Mensaje, Titulación no encontrada

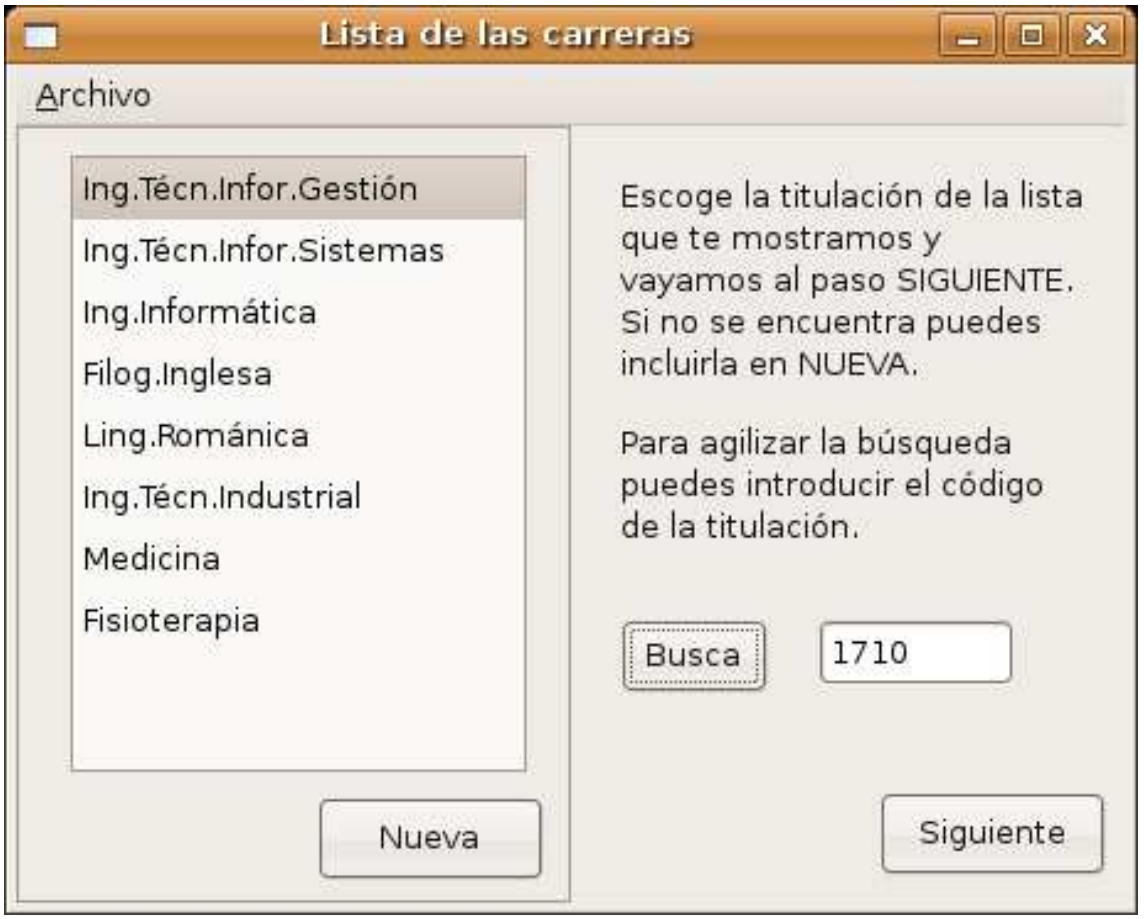

Figura 7.7: Mensaje, Titulación encontrada

Para pasar al siguiente paso pulsaremos “Siguiente” para seguir con la pantalla “Algoritmos”, el sistema antes de seguir comprobará si efectivamente tenemos una titulación seleccionada, en caso contrario se nos mostrará el siguiente mensaje:

Figura 7.8: Mensaje, Titulación no seleccionada

7.1.5. ¡Alguien nuevo!

¡Hemos decidido introducir una nueva titulación! Para dar este paso anteriormente se nos ha mostrado el mensaje de advertencia siguiente:

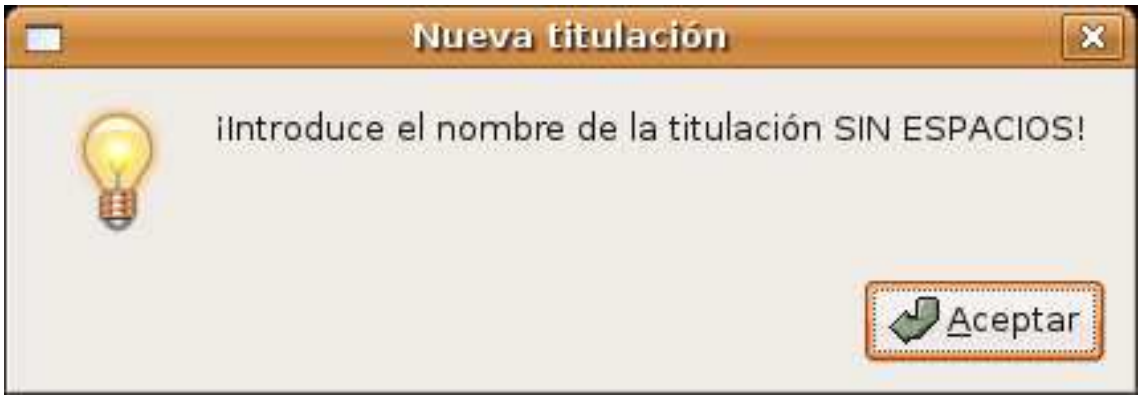

Figura 7.9: Mensaje, ¡Sin espacios!

Seguidamente se nos muestra la pantalla para introducir una nueva Titulación, en la que si nos arrepentimos, podemos pulsar el botón "Cerrar" que nos aparece en la pantalla de tal forma que volveremos a la pantalla de "Lista de Titulaciones" sin mostrar ninguna modificación.

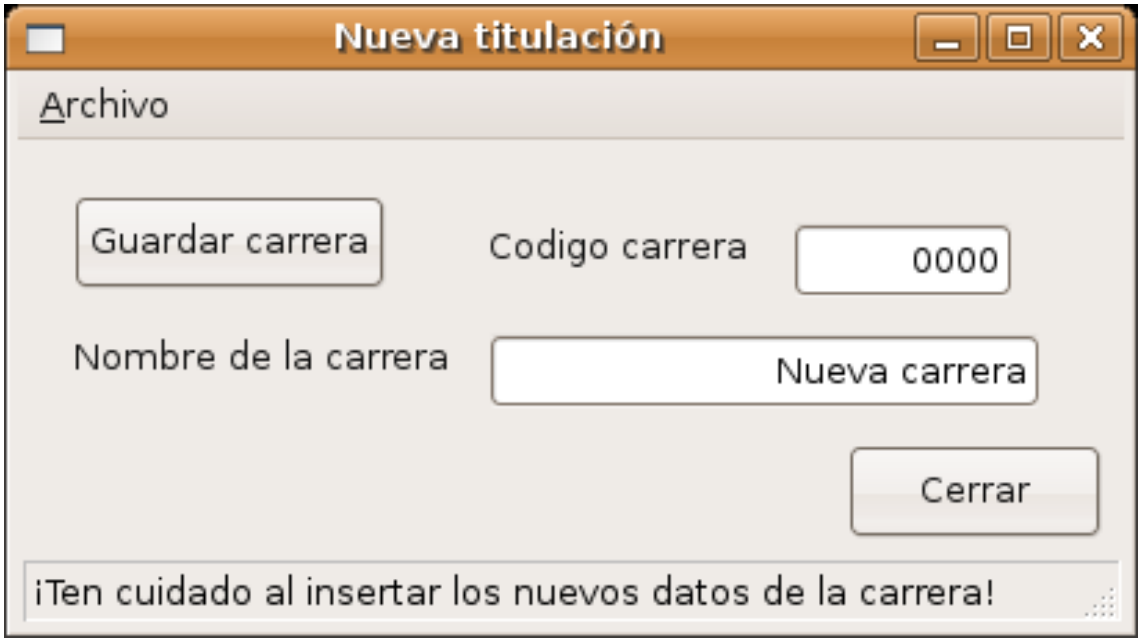

Figura 7.10: Pantalla nueva Titulación

En el caso de que queramos continuar con el proceso, tendremos que introducir el código de la misma y su nombre como ya hemos comentado, sin espacios; podremos abreviar los nombres quedando unidos mediante puntos o bien podremos unirlos mendiante un guión bajo entre las palabras, por ejemplo: Ciencias_del_Mar.

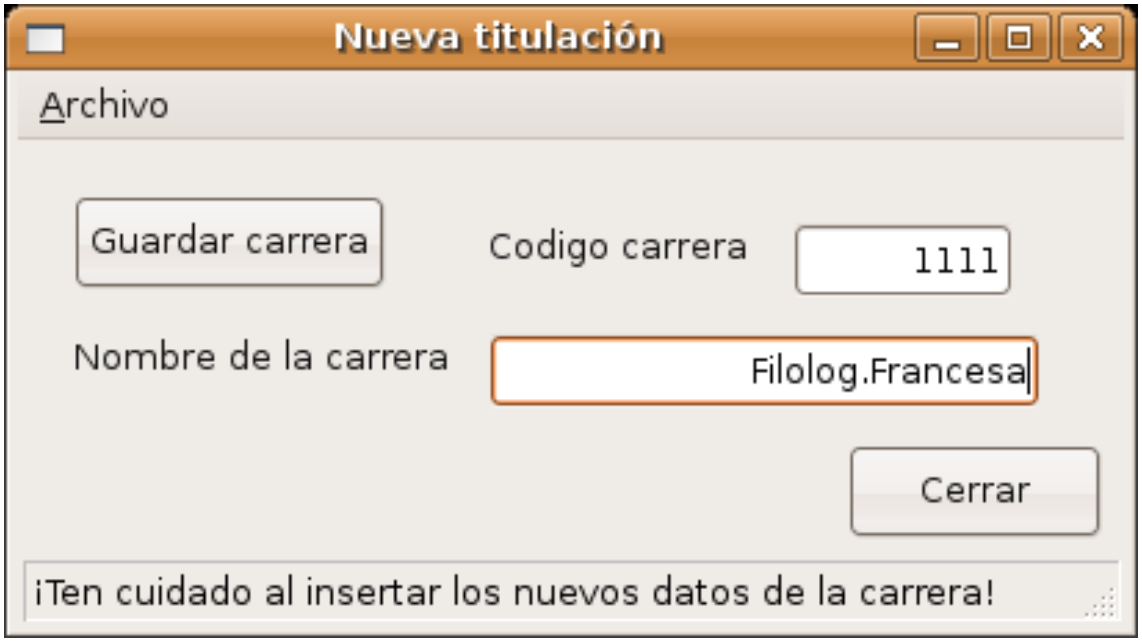

Figura 7.11: Introducimos nueva Titulación

Finalmente pulsando el botón “Guardar carrera” se nos quedará registrada la carrera y podremos utilizarla para realizar nuestro calendario. Seguidamente se nos muestra la pantalla de “Lista de Titulaciones” ya actualizada.

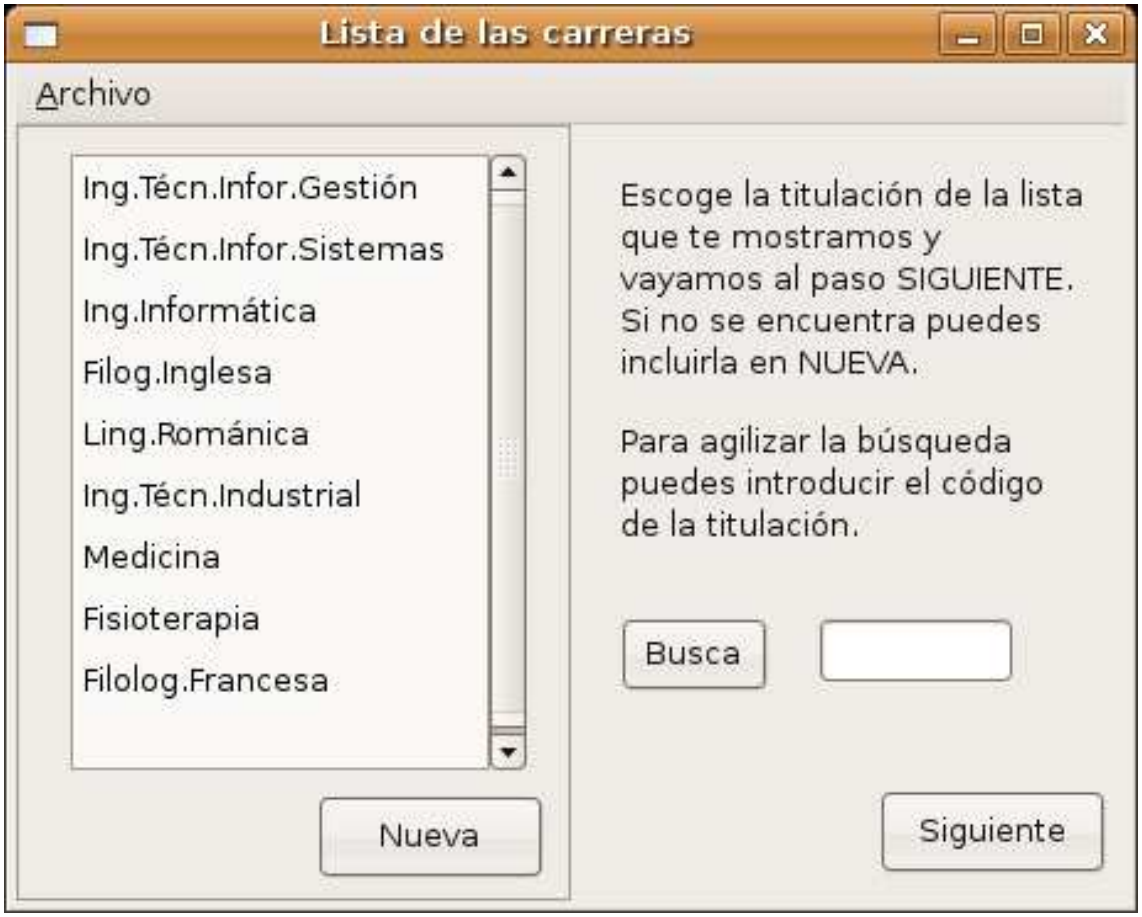

Figura 7.12: Pantalla Lista de Titulaciones actualizada

7.1.6. Los tres algoritmos

Antes de mostrar esta pantalla, tenemos que darle tiempo al programa (sólo son 10 segundos) en los que la aplicación está procesando los datos que le hemos indicado que vamos a utilizar. Se nos muestra el siguiente mensaje:

Figura 7.13: Procesamiento de datos

Sólo estamos a un paso de conseguir nuestro calendario. En esta pantalla se nos muestra una pantalla en la que inicialmente nos dice las diferentes opciones que tenemos para navegar por ella. Pulsando sobre los diferentes botones de "Algoritmo 1", "Algoritmo 2" y "Algoritmo 3", podremos conocer de una forma breve el funcionamiento de los diferentes algoritmos y una descripción de los mismos.

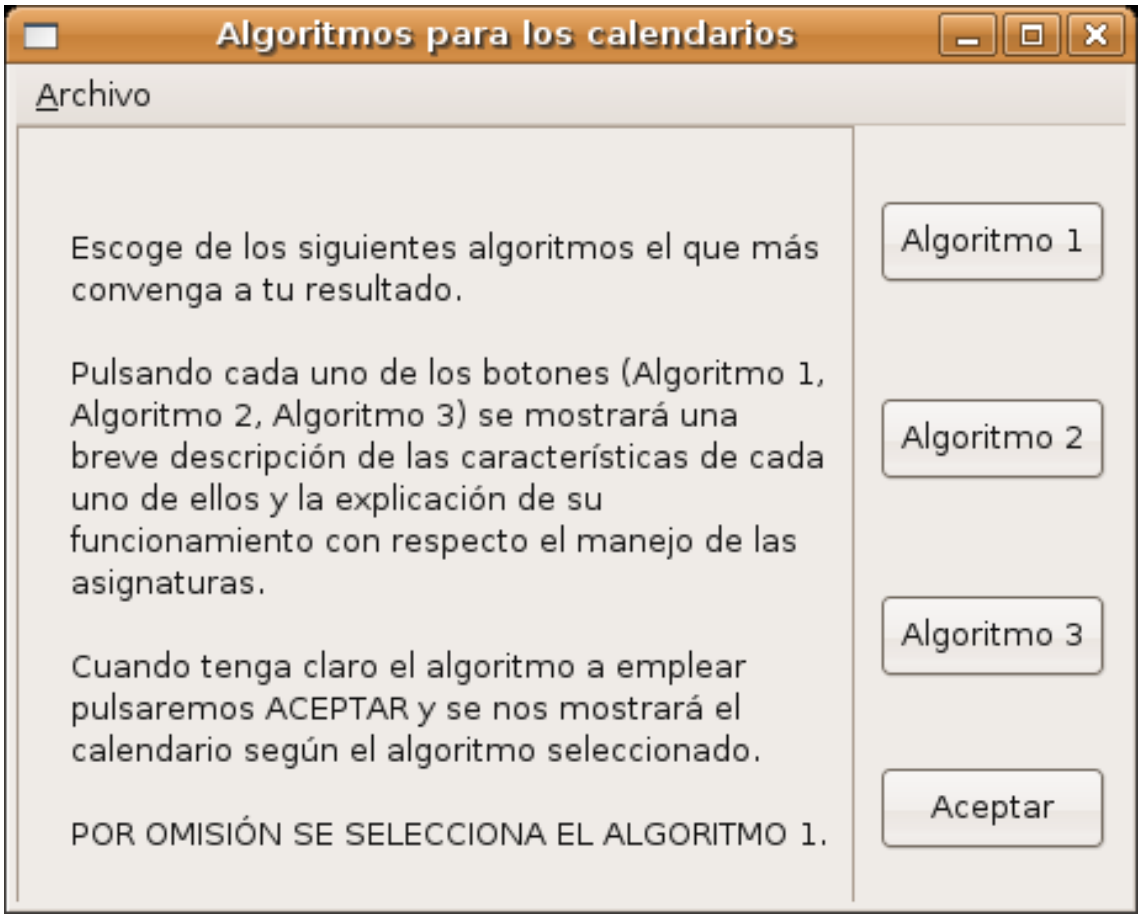

Figura 7.14: Pantalla inicial de Algoritmos

Pulsando el botón de “Algoritmo 1”, Algoritmo Devorador, se nos muestra la siguiente información:

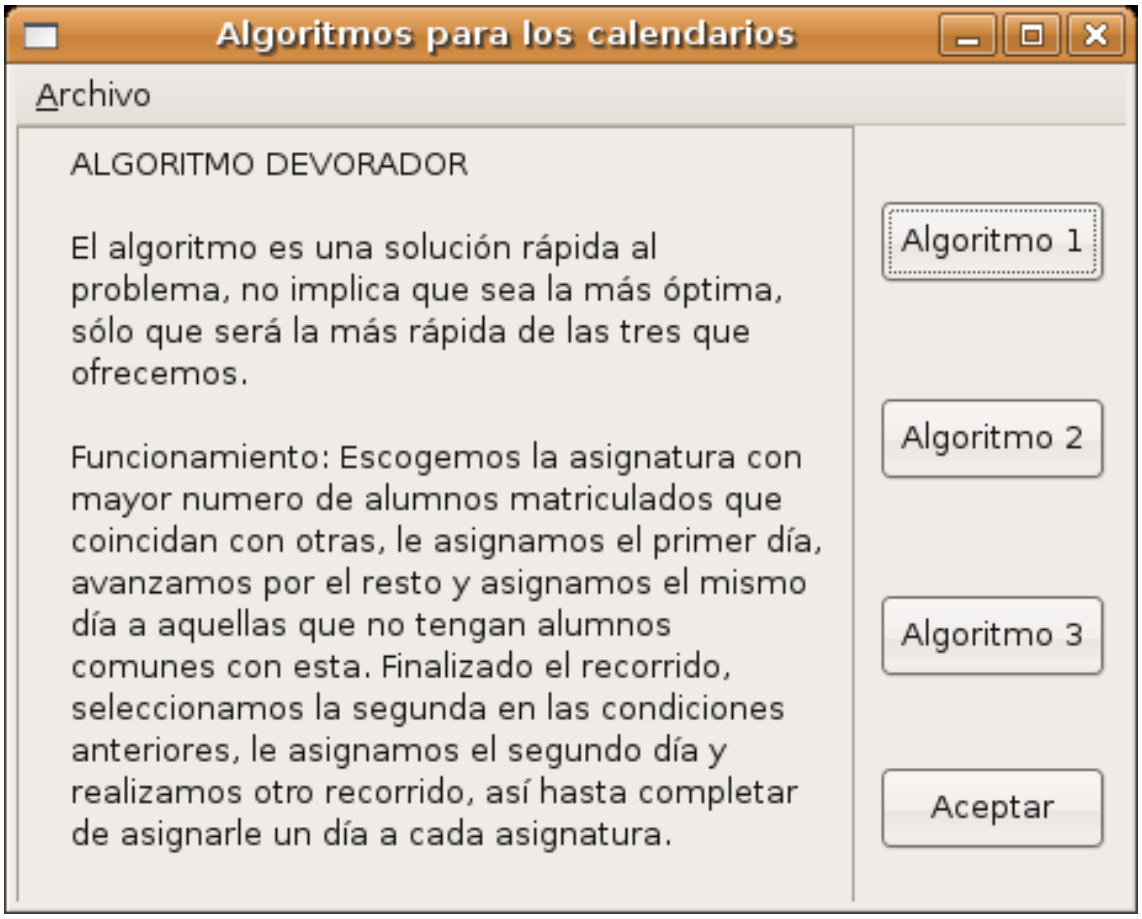

Figura 7.15: Algoritmo 1, Devorador

Pulsando el botón de “Algoritmo 2”, Algoritmo BackTracking, se nos muestra la siguiente información:

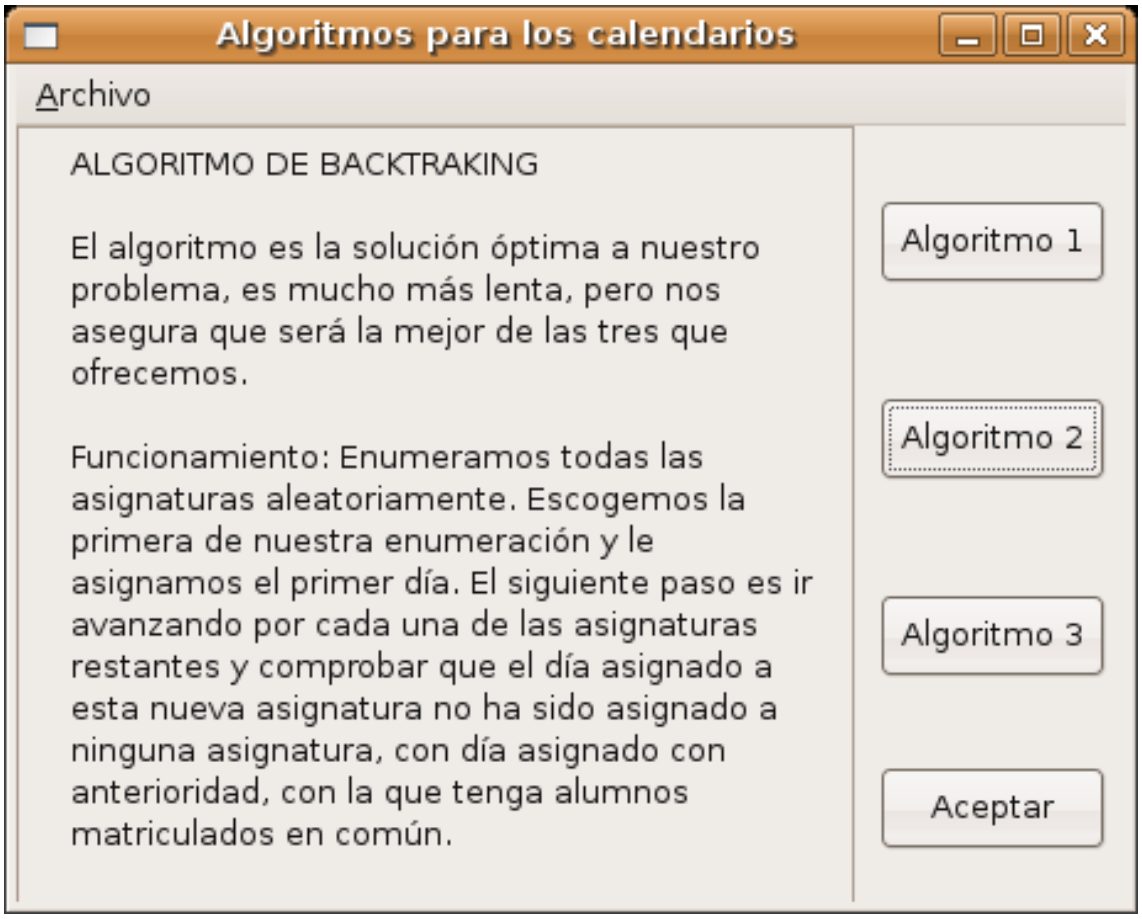

Figura 7.16: Algoritmo 2, BackTracking

Pulsando el botón de “Algoritmo 3”, Algoritmo Devorador Mejorado, se nos muestra la siguiente información:

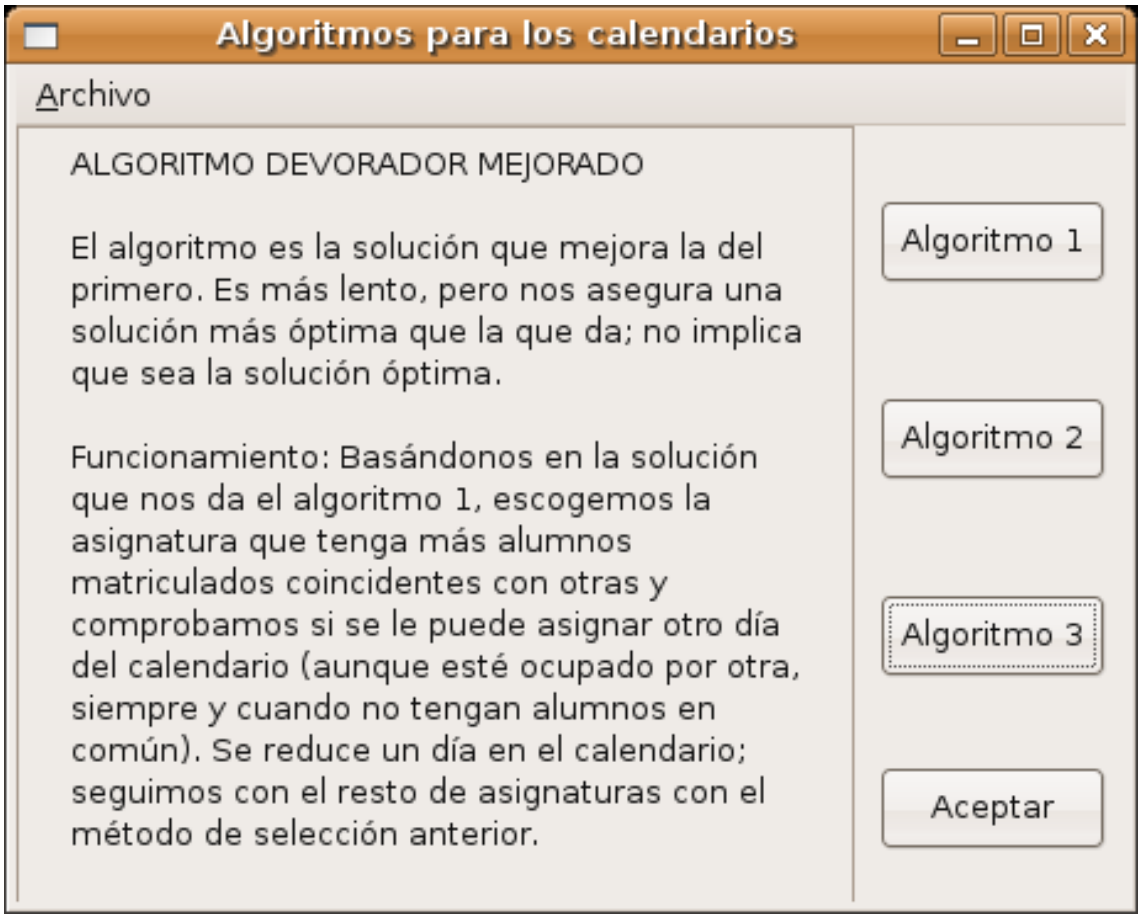

Figura 7.17: Algoritmo 3, Devorador Mejorado

Según el algoritmo que tengamos en pantalla, será el algoritmo que hayamos seleccionado. Si es el que visualizamos el que queremos, pulsaremos el botón “Aceptar” que aparece en pantalla. Si directamente de la información de inicio de esta pantalla pulsamos “Aceptar” como bien viene indicado, se selecciona por omisión el Algoritmo 1 Devorador.

7.1.7. Voilá!

Según el algoritmo escogido, se nos mostrará un mensaje indicándonos que ya se ha finalizado de hacer los cálculos; y seguidamente se nos mostrará el resultado.

Figura 7.18: Cálculo terminado del Algoritmo 1, Devorador

Figura 7.19: Cálculo terminado del Algoritmo 2, BackTracking

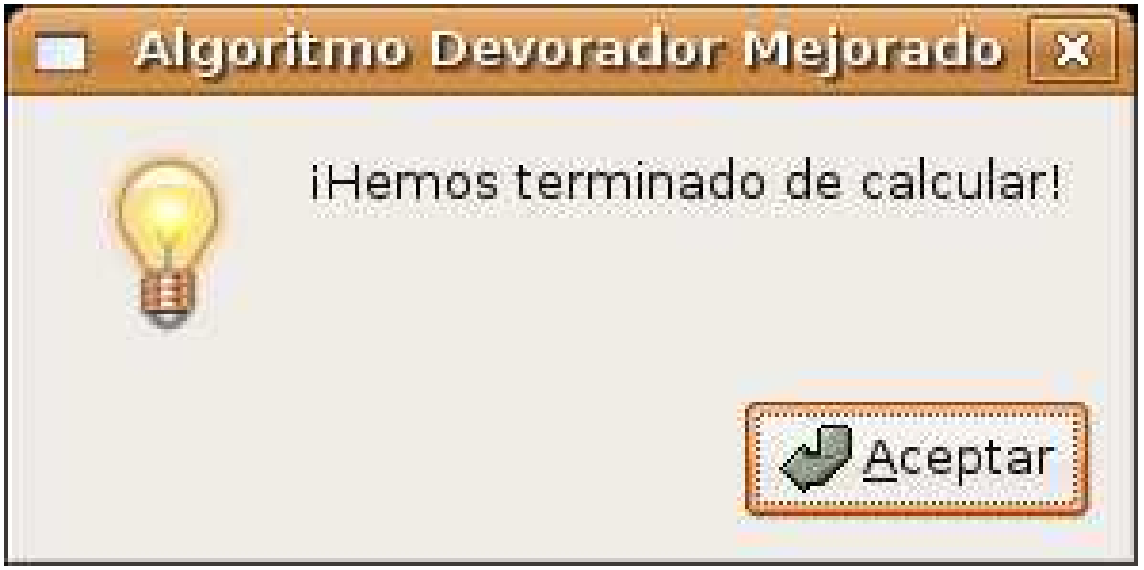

Figura 7.20: Cálculo terminado del Algoritmo 3, Devorador Mejorado

En la pantalla Resultado se nos muestra una tabla con la siguiente información: código de la asignatura, nombre de la asignatura, fecha asignada para realizar el examen de dicha asignatura, hora en la que el examen tendrá lugar en dicha fecha.

	A	B	C	D
1	1710043	PROGRAMACIÓN EN INTERNET	3/12/2007	10:00
2	1710032	INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL	3/12/2007	17:00
3	1710034	PROGRAMACIÓN CONCURRENTE Y DISTRIBUIDA	3/12/2007	17:00
4	1710026	TRADUCTORES	4/12/2007	10:00
5	1710031	HERRAMIENTAS DE DESARROLLO RÁPIDO DE APLICACIONES	4/12/2007	10:00
6	1710024	BASES DE DATOS I	4/12/2007	17:00
7	1710029	TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES	4/12/2007	17:00
8	1710022	PROGRAMACIÓN ORIENTADA A OBJETOS	5/12/2007	10:00
9	1710025	BASES DE DATOS II	5/12/2007	10:00
10	1710019	INTRODUCCIÓN A LA PROGRAMACIÓN	5/12/2007	17:00
11	1710021	INFORMÁTICA GENERAL	5/12/2007	17:00
12	1710017	INGENIERÍA DEL SOFTWARE I	6/12/2007	10:00
13	1710018	INGENIERÍA DEL SOFTWARE II	6/12/2007	10:00

Figura 7.21: Calendario, solución

Si queremos volver a la pantalla de los “Algoritmos” sólo tendremos que pulsar en el botón “Atrás” que aparece en pantalla. Y tendremos la posibilidad de nuevo de realizar nuestro calendario con otro algoritmo. Si queremos imprimir nuestro calendario, lo podremos hacer pulsando en el botón “Imprimir” que se nos aparece, si lo hacemos, se nos mostrará un mensaje indicándonos la ubicación de nuestro PDF con el resultado que se nos muestra en pantalla:

Figura 7.22: Mensaje de impresión

Si nos dirigimos al directorio que se nos indica, encontraremos un PDF que contendrá nuestra solución:

CALENDARIO - Titulación: Ing.Técn.Infor.Gestión

CODIGO	NOMBRE	FECHA	HORA
1710043	PROGRAMACIÓN EN INTERNET	3/12/2007	10:00
1710032	INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL	3/12/2007	17:00
1710034	PROGRAMACIÓN CONCURRENTE Y DISTRIBUIDA	3/12/2007	17:00
1710026	TRADUCTORES	4/12/2007	10:00
1710031	HERRAMIENTAS DE DESARROLLO RÁPIDO DE APLICACIONES	4/12/2007	10:00
1710024	BASES DE DATOS I	4/12/2007	17:00
1710029	TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES	4/12/2007	17:00
1710022	PROGRAMACIÓN ORIENTADA A OBJETOS	5/12/2007	10:00
1710025	BASES DE DATOS II	5/12/2007	10:00
1710019	INTRODUCCIÓN A LA PROGRAMACIÓN	5/12/2007	17:00
1710021	INFORMÁTICA GENERAL	5/12/2007	17:00
1710017	INGENIERÍA DEL SOFTWARE I	6/12/2007	10:00
1710018	INGENIERÍA DEL SOFTWARE II	6/12/2007	10:00
1710015	SISTEMAS OPERATIVOS I	6/12/2007	17:00
1710016	SISTEMAS OPERATIVOS II	6/12/2007	17:00
1710008	ANÁLISIS Y DISEÑO DE ALGORITMOS I	7/12/2007	10:00
1710009	ANÁLISIS Y DISEÑO DE ALGORITMOS II	7/12/2007	10:00
1710005	ESTRUCTURA DE DATOS I	7/12/2007	17:00
1710006	ESTRUCTURA DE DATOS II	7/12/2007	17:00
1710007	METODOLOGÍA DE LA PROGRAMACIÓN	7/12/2007	17:00

Figura 7.23: Calendario, solución PDF

7.1.8. Help!, ¡Ayuda!, SOS

Hemos dejado para el final del manual la pantalla de ayuda ya que ésta es bastante intuitiva y la podemos solitar en cualquiera de las pantallas. Si no aparece ningún botón en el entorno de la misma, siempre podemos encontrarla dentro del menú de "Archivo".

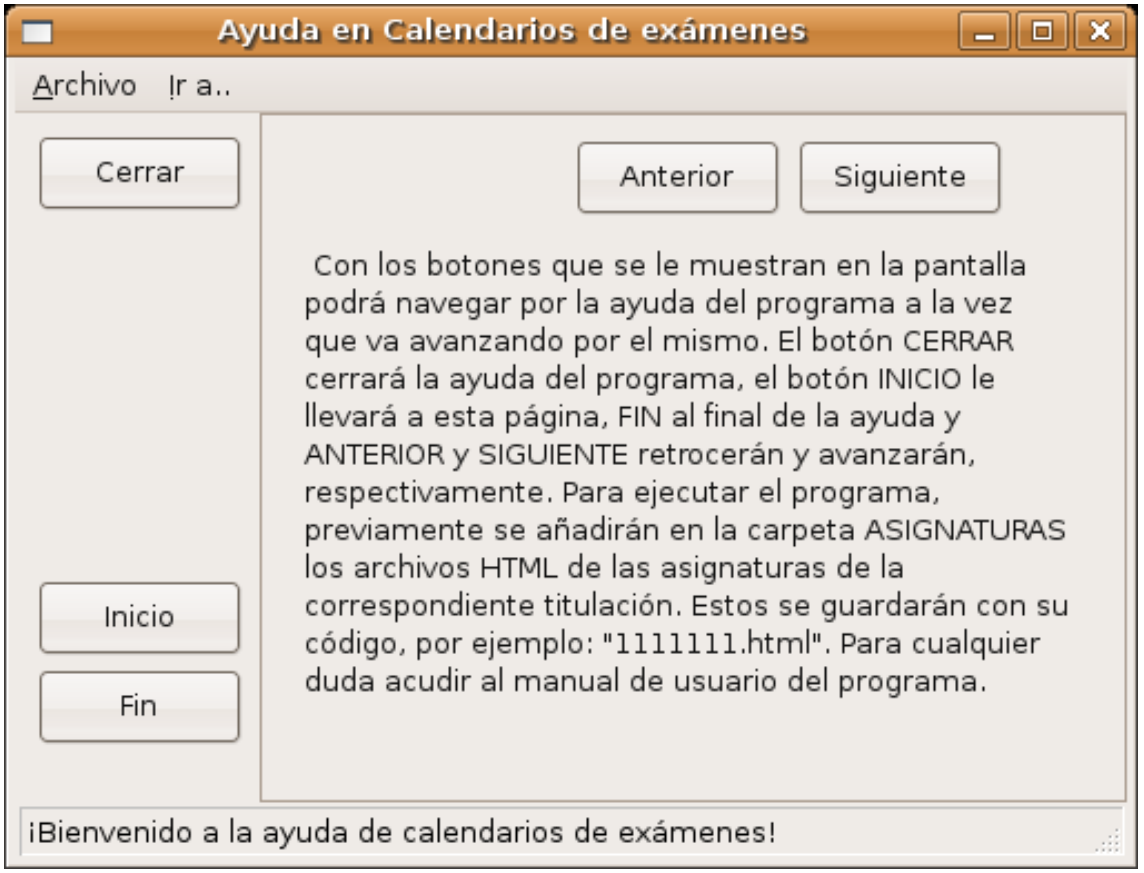

Figura 7.24: Calendario, solución

Ésta según el paso en el que nos encontremos en el programa, nos mostrará una información u otra. A pesar de esto, podemos navegar dentro de la ayuda sin necesidad de estar en un paso u otro del programa, pulsaremos "Inicio" para mostrar la información de presentación de la ayuda, "Siguiente" para visualizar la información de la siguiente pantalla en la que nos situemos, "Fin" para llegar al final de la ayuda y "Anterior" para mostrar la información de la pantalla anterior de la que nos encontremos. Todos estas operaciones también se añaden dentro del menú "Ir a" que se muestra en la ayuda. Si lo deseamos, podemos cerrar la ayuda pulsando "Cerrar" y seguiremos ejecutando nuestro programa inicial (obviamente podemos manejarlo y ejecutar nuestro programa inicial a la vez que tenemos abierta la ayuda del mismo).

Capítulo 8

Manual de Instalación

8.1. Instalando wxWidgets

Como ya hemos comentado, uno de los lenguajes que empleamos en nuestra aplicación son las librerías wxWidgets, en este apartado vamos a indicar cómo encontrarlas y cómo instalarlas en nuestro sistema.

Como ya hemos comentado en apartados anteriores, las librerías wxWidgest son soportadas por todos los sistemas operativos sin ningún problema, pero debido a que en nuestra aplicación utilizamos "scripts", no podemos hacer que el programa se ejecute en todos los sistemas operativos, de ahí a que sólo expliquemos la instalación de las librerías en un entorno Linux.

La búsqueda de estas librerías es bastante simple, sólo tendremos que buscar de forma avanzada dentro de nuestro programa de "Añadir y quitar aplicaciones" (Synaptic o Kynaptic) las librerías "wxWidgets" nos aparecerán en pantalla todos los paquetes relacionados con estas librerías. Sólo tendremos que añadir los siguientes (los paquetes necesarios para estos serán incluidos por la aplicación indicándonoslo mediante un mensaje):

- wx2.8-headers
- libwxgtk2.8-dev
- libwxgtk2.8-0
- libwxbase2.8-dev
- libwxbase2.8-0

Obviamente nos instalaremos la última versión de estas librerías, aunque en el caso de la versión anterior la "2.6" tampoco hay muchas diferencias son sólo algunas funciones, que en nuestro programa no empleamos, que han añadido.

8.2. Instalando recode

La instrucción recode dentro de las operaciones del sistema operativo, nos sirve para la codificación de los ficheros que se nos crean a lo largo de la aplicación. Queremos que estos tengan una codificación UTF-8 para que se muestren correctamente todos los acentos y símbolos de los ficheros generados y con esta operación lo conseguimos facilmente. La forma de bajarnos esta aplicación es la siguiente:

apt-get install recode

Con esta instrucción escribiéndola en consola se nos descargarán todos los paquetes que necesita la aplicación y podremos recodificar los ficheros sin problemas. Si no tenemos permiso para ejecutarla, tendremos que realizarlo como administrador y tendríamos que escribir entonces lo siguiente:

sudo -s apt-get install recode

Nos pedirían la constraseña del superusuario y seguidamente se instalarían todos los paquetes que le sean necesarios a la aplicación.

8.3. Instalando LaTeX

Hay que instalar el paquete tetex. Éste incluye todo lo necesario, excepto el editor para poder escribir los documentos LaTeX. El paquete tetex lo podemos encontrar en la aplicación de "Agregar y quitar programas" que tenga nuestro sistema operativo Linux. Esto nos servirá para poder imprimir el resultado final del programa en PDF, ya que generamos el fichero LaTeX con los resultados obtenidos y finalmente lo pasamos a PDF.

En nuestro caso no necesitaríamos un editor, pero ya que se ha mencionado antes, podemos emplear los siguientes: "Emacs" (paquete emacs), "LyX" (paquete lyx)... entre otros

Capítulo 9

Referencias

Aquí una lista de referencias bibliográficas consultadas. Así mismo se incluyen las referencias a documentos con acceso en el World Wide Web consultados:

a) Kennt H.Rosen. "Discrete Mathematics and its Applications"

b) Steven S. Skiena. "The algorithm design manual Telos"

c) Martí, M.; Ortega, Y.; Verdejo, A.: "Estructuras de datos y métodos algorítmicos: ejercicios resueltos". Prentice Hall, 2004.

d) Ellie Quigley "PERL by Example"

e) Stevens, Richard W. "Advanced Programming in the Unix Environment". Reading, Masachusetts: Addison-Wesley, 1992.

f) `http://www.geom.uiuc.edu/~zarembe/graph9.html`

g) `http://www.wxwidgets.org`

h) `http://es.wikipedia.org/wiki/WxWidgets`

i) `http://www.wxwidgets.org/docs/book/`

j) `http://www.wxformbuilder.org/`

k) `http://www.dma.fi.upm.es/grafos/color03.PDF`

l) `http://www.ieev.uma.es/matdisc/a-grafos.pdf`

m) `http://www.programacion.com/articulo/grafos_asignacion/`

n) `http://mipagina.cantv.net/jhnieto1/c12.pdf`

ñ) `http://mat.gsia.cmu.edu/COLOR/color.html`

o) `http://web.cs.ualberta.ca/~joe/Coloring/`

p) `http://decsai.ugr.es/~mgs/ed/stringstram.pdf`

q) `http://www.cim.mcgill.ca/~wahab/graphtheory/cgwebpage/algorithm.html`

r) `http://www.wxwidgets.org/manuals/stable/wx_contents.htmlhttp://homepages.cwi.nl/~daybuild/daily-docs/toolbus/dc/dad/perl-adapter_8c-source.html`

s) `http://es.wikipedia.org/wiki/GNU_GPL`

Capítulo 10

Anexos

10.1. Licencia GPL

10.1.1. GNU GPL

GNU GPL (Licencia Pública General) es una licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

Existen varias licencias "hermanas"de la GPL, como la licencia de documentación libre GNU (GFDL) que cubre los artículos de la Wikipedia, la Open Audio License, para trabajos musicales, etcétera, y otras menos restrictivas, como la MGPL, o la LGPL (Lesser General Public License o Library General Public License), que permiten el enlace dinámico de aplicaciones libres a aplicaciones no libres.

10.1.2. Validez legal

La licencia GPL, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se le denomina contrato de licencia o acuerdo de licencia. En los países de tradición anglosajona existe una distinción doctrinal entre licencias y contratos, pero esto no ocurre en los países de tradición civil o continental. Como contrato, la GPL debe cumplir los requisitos legales de formación contractual en cada jurisdicción.

La licencia ha sido reconocida por juzgados en Alemania, particularmente en el caso de una sentencia en un tribunal de Munich, lo que indica positivamente su validez en jurisdicciones de derecho civil.

Dentro de varios marcos regulatorios y legislaciones de países alrededor del mundo, la inserción y utilización de la GPL puede encontrar dificultades desde el punto de vista de la aplicación y validez, debido a la rigidez o contravención de leyes y normas en defensa de los derechos de los consumidores. Sin embargo, se

puede aducir que la GPL no es un contrato del consumidor ya que el licenciatario por lo general no asume las características de un negocio.

10.2. GPL

Esta es la traducción al español de la famosa licencia GPL "GNU Public License"(GPL), versión 2 que cubre la mayor parte del software de la Free Software Foundation, y muchos más programas.

IMPORTANTE: Esta traducción tiene carácter méramente informativo y carece de validez legal. Si distribuye software libre o ha recibido software libre con esta traducción utilice para cuestiones legales siempre la GPL versión 2 en Ingles de la Free Software Foundation.

10.2.1. GNU GENERAL PUBLIC LICENSE (GPL)

Versión 2, Junio 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Está permitido copiar y distribuir copias idénticas de esta

licencia, pero no está permitida su modificación.

10.2.2. *Preámbulo*

Las licencias de la mayoría del software están diseñadas para eliminar su libertad de compartir y modificar dicho software. Por contra, la GNU General Public License (GPL) está diseñada para garantizar su libertad de compartir y modificar el software. Software libre para garantizar la libertad de sus usuarios. Esta licencia GNU General Public License (GPL) se aplica en la mayoría de los programas realizado por la Free Software Foundation (FSF, Fundación del Software Libre) y en cualquier otro programa en los que los autores quieran aplicarla. También, muchos otros programas de la Free Software Foundation están cubiertos por la GNU Lesser General Public License (LGPL) e igualmente puede usarla para cubrir sus programas.

Cuando hablamos de Software Libre, hablamos de libertad, no de precio. Nuestra licencia General Public License (GPL) está diseñada para asegurarle las libertades de distribuir cópias de Software Libre (y cobrar por ese servicio si quiere), asegurarle que recibirá el código fuente del programa o bien podrá conseguirlo si quiere, asegurarle que puede modificar el programa o modificar algunas de sus piezas para un nuevo programa y para garantizarle que puede hacer todas estas cosas.

Para proteger sus derechos, necesitamos realizar restricciones que prohíben a cualquiera denegar estos derechos o pedirle que reniegue de sus derechos. Estas restricciones se traducen en ciertas obligaciones por su parte si ustéd piensa distribuir copias del programa o tiene intención de modificarlo.

Por ejemplo, si usted distribuye copias de un programa, ya sea gratuitamente o no, usted tiene que otorgar a sus clientes todos los derechos que ha adquirido usted con el programa. Usted tiene que asegurarse de que sus clientes reciben o pueden recibir el código fuente si lo solicitan y ustéd tiene que mostrarles los términos de la licencia para que conozcan sus derechos.

Nosotros protegemos los derechos con dos pasos:

a) utilizando el copyright en nuestros programas, y

b) ofreciéndole esta licencia que le garantiza legalmente el derecho a copia, redistribución y/o modificación de este programa.

Además, para nuestra protección y la de cada uno de los autores, queremos tener la certeza de que todo el mundo entiende que no hay ninguna garantía por este Software Libre. Si el programa es modificado por alguien más y este falla, ese software no es el original y por tanto cualquier problema introducido por otras personas no afecta a la reputación de los autores originales.

Por último, cualquier programa está amenazado constantemente por las patentes de software. Desearíamos evitar el riesgo que distribuidores de programas libres adquieran individualmente patentes, transformando de facto el software libre en privativo. Para evitar este problema, dejamos claro que cualquier patente deberá ser licenciada para permitir el uso libre de cualquier persona o no ser patentada.

Los términos y condiciones para la copia, distribución y modificación del software se especifican en el siguiente punto.

10.2.3. TÉRMINOS Y CONDICIONES PARA LA COPIA, DISTRIBUCIÓN Y MODIFICACIÓN

a) La licencia se aplica a cualquier programa u otro trabajo que contenga un aviso del dueño del copyright diciendo que el software debe ser distribuido bajo los términos y condiciones de esta licencia General Public License (GPL). El "programa", desde ahora, se refiere tanto a el Programa como a cualquier trabajo derivado bajo la ley del copyrigth: es decir, un trabajo que contenga el programa o una porción de el, tanto copia como con modificaciones y/o traducido a otra lengua (más abajo, la traducción está incluida sin limitaciones en el término "Modificación".) Cada licencia está asignada a usted.

Otras actividades distintas de la copia, distribución y modificación no están cubiertas por esta licencia y están fuera de su objetivo. Ejecutar el programa no está restringido, y el resultado del programa está cubierto por la licencia solo si su contenido contribuye parte de un trabajo derivado del

Programa (independiente de la ejecución del programa). Incluso si esto es verdad, depende de lo que haga el programa

b) Ustéd puede copiar y distribuir copias exactas del código fuente del Programa tal y como lo recibió, usando cualquier medio, a condición de, adecuadamente y de forma bien visible, publique en cada copia una nota de copyright y un repudio de garantía; mantenga intactas todas las notas que se refieran a esta licencia y a la exención de garantía; y proporcione a los receptores del Programa una copia de esta Licencia junto al programa.

Usted puede cobrar unos honorarios por la transferencia física de la copia, y puede a su criterio ofrecer una garantía adicional por un precio.

c) Ustéd puede modificar su copia o copias del Programa o cualquier porción de ella, obteniendo así un trabajo derivado del Programa, y copiar y distribuir estas modificaciones o trabajo derivado bajo los mismos término de la Sección 1, antedicho, cumpliendo además las siguientes condiciones:

1) Debe hacer que los ficheros modificados contengan información visible de que ha modificado el fichero y la fecha de cualquier cambios.
2) Debe hacer que cualquier trabajo que distribuya o publique y que en su totalidad o en parte contenga o sea derivado del Programa o de cualquier parte de el, sea licenciado como un todo, sin carga alguna a las terceras partes, bajo los términos de esta licencia.
3) Si la modificación del programa normalmente interpreta comandos interactivos en su ejecución, debe, cuando comience su ejecución para ese uso interactivo de la forma más habitual, imprimir o mostrar un aviso de exención de garantía (o por el contrario que sí ofrece garantía) y de como los usuarios pueden redistribuir el programa bajo estas condiciones, e informando a los usuarios de como pueden obtener una copia de esta Licencia. (Excepción: Si el programa es interactivo pero normalmente no muestra este anuncio, no es necesario en un trabajo derivado mostrar este aviso).

Estos requisitos se aplican a las modificaciones como un todo. Si secciones identificables del trabajo no están derivadas del Programa, pueden ser razonablemente consideradas independientes y trabajos separados en si mismos, por tanto esta Licencia, y sus términos, no se aplican a estas secciones cuando ustéd las distribuye como trabajos independientes. Pero cuando usted distribuye las mismas secciones como parte de un todo que es un trabajo derivado del Programa, la distribución del todo debe respetar los términos de esta licencia, cuyos permisos para otros licenciatarios se extienden al todo, y por lo tanto a todas y cada una de sus partes, con independencia de quién la escribió.

Por lo tanto, no es la intención de esta sección reclamar derechos o desafiar sus derechos sobre trabajos escritos totalmente por usted mismo. Por el contrario, la intención es ejercer el derecho a controlar la distribución de trabajos derivados o colectivos basados en el Programa.

Además, el mero acto de agregar otro trabajo no basado en el Programa con el Programa (o con otro trabajo derivado del Programa) en un

volumen de almacenamiento o un medio de distribución no consigue que el otro trabajo se encuentre bajo los términos de esta licencia.

d) Ustéd puede modificar su copia y distribuir el Programa (o un trabajo derivado, cubierto bajo la Sección 2) en formato objeto o ejecutable bajo los términos de las Secciones 1 y 2 antedichas proporcionado con el al menos una de las siguientes:

1) Acompañando el Programa con con el código fuente completo, legible por un ordenador, correspondiente a la arquitectura correspondiente, que debe ser distribuido bajo los términos de las secciones 1 y 2 usando un médio físico habitual en el intercambio de software; o,
2) Acompañando el Programa con una oferta por escrito, valida al menos por tres años, de facilitar a cualquier tercera parte, sin un cargo mayo del coste del médio físico, una copia completa legible por un ordenador del código fuente de la arquitectura elegida, que será distribuido bajo los términos de las Secciónes 1 y 2 usando un médio físico habitual en el intercambio de software; o,
3) Acompañando el Programa con un la información que recibió, ofreciendo distribuir el código fuente correspondiente. (Esta opción se permite sólo para distribuir software gratuito -no comercial- y sólo si recibió el programa como código objeto o en formato ejecutable con esta misma oferta, de acuerdo con el apartado b anterior).

por código fuente de un trabajo se entiende la forma óptima para realizar modificaciones en el. Para un programa ejecutable, el código fuente completo se refiere a todo el código fuente para todos los módulos que contiene, mas cualquier fichero asociado de definición de interfaces, mas el script utilizado para la compilación y la instalación del ejecutable. Sin embargo, como una excepción especial, el código fuente distribuido no necesita incluir nada que no sea normalmente distribuido (ni en código fuente o en forma binaria) con los componentes mas importantes (compiladores, kernels y demás) del sistema operativo donde corre el ejecutable, salvo que el componente acompañe al ejecutable.

Si la distribución de ejecutables o compilado se realiza ofreciendo acceso a un sitio para la copia, entonces ofrecer un acceso equivalente de copia desde un sitio para el código fuente cuenta como una distribución de código fuente, incluso aunque terceras partes no estén obligadas a copiar el código fuente con el código compilado.

e) Usted no debe copiar, modificar, sublicenciar o distribuir el Programa excepto como está permitido expresamente en esta Licencia. Cualquier intento de copiar, modificar, sublicenciar o distribuir el Programa que no esté incluido en la Licencia está prohibido, y anulará automáticamente los derechos otorgados por esta licencia. Sin embargo, las partes que hayan recibido copias, o derechos, por usted bajo esta Licencia no verán sus licencias terminadas mientras estas partes continúen cumpliendo los términos de esta licencia.

f) Usted no está obligado a aceptar esta licencia, ya que usted no la ha firmado. Sin embargo, nada mas le garantiza los derechos de modificación o

distribución del programa o de sus trabajos derivados. Estas acciones están prohibidas por la ley si usted no acepta esta Licencia. En cualquier caso, por modificar o distribuir el programa (o cualquier trabajo derivado del programa), usted indica su aceptación implícita de esta Licencia, ya que la necesita para hacerlo, y todos sus términos y condiciones de copia, distribución o modificación del Programa o trabajos derivados.

g) Cada vez que usted redistribuya el Programa (o cualquier trabajo derivado del Programa), el receptor automáticamente recibe la licencia por parte del licenciatario original para copiar, distribuir o modificar el Programa sujeto a estos términos y condiciones. Usted no puede imponer ninguna otra restricción a los receptores limitando los derechos garantizados en esta Licencia. Usted no es responsable de asegurar el cumplimiento de terceras partes sobre la Licencia.

h) Si, como consecuencia de una decisión judicial o una acusación de infracción de patentes o por cualquier otra razón (no limitada a una causa de patentes), le son impuestas condiciones (ya sea por una orden judicial, por un acuerdo o cualquier otra forma) que contradiga los términos y condiciones d esta Licencia, no le exime de cumplir los términos y condiciones de dicha Licencia. Si usted no puede distribuir el Programa cumpliendo simultáneamente tanto los términos y condiciones de la Licencia como cualquier otra obligación que le haya sido impuesta, usted consecuentemente no puede distribuir el Programa bajo ninguna forma. Por ejemplo, si una patente no permite la redistribución gratuita del Programa por parte de todos aquellos que reciben copias directa o indirectamente a través de usted, entonces la única forma de satisfacer tanto esa condición como los términos y condiciones de esta Licencia sería evitar completamente la distribución del Programa.

Si alguna porción de esta sección es inválida o imposible de cumplir bajo una circunstancia particular, el resto de la sección tiene que intentar aplicarse y la seccionó completa debe aplicarse en cualquier otra circunstancia.

El propósito de esta sección no es inducir a infringir ninguna patente o otros derechos de propiedad o impugnar la validez de estos derechos; esta sección tiene le único propósito de proteger la integridad del sistema de distribución del Software Libre, que está implementado bajo practicas de licencias públicas. Mucha gente ha realizado generosas contribuciones a la gran variedad de software distribuido bajo este sistema con confianza en una aplicación consistente del sistema; será el autor/donante quien decida si quiere distribuir software mediante cualquier otro sistema y una licencia no puede imponer esa elección.

Este apartado pretende dejar completamente claro lo que se cree que es una consecuencia del resto de esta Licencia.

i) Si la distribución y/o el uso del Programa está restringido en ciertos países, ya sea por patentes o por interfaces bajo copyright, el propietario del Copyright original que pone el Programa bajo esta Licencia debe añadir unos límites geográficos expecíficos excluyendo esos países, por lo que la distribución solo estará permitida en los países no excluidos. En este caso, la

Licencia incorpora la limitación de escribir en el cuerpo de esta Licencia.

j) La Free Software Foundation puede publicar versiones revisadas y/o nuevas de la Licencia GPL de cuando en cuando. Dichas versiones serán similares en espíritu a la presente versión, pero pueden ser diferentes en detalles para considerar nuevos problemas o situaciones.

Cada versión tiene un número de versión propio. Si el Programa especifica un número de versión de esta Licencia que hace referencia a esta o çualquier versión posterior", usted tiene la opción de seguir los términos y condiciones de o bien la versión referenciada o bien cualquiera de las versiones posteriores publicadas por la Free Software Foundation. Si el Programa no especifica ningún número de versión, usted debe elegir cualquiera de las versiones publicadas por la Free Software Foundation.

k) Si usted quiere incorporar parte del Programa en otros programas libres cuyas condiciones de distribución son diferentes, escriba al autor para pedir permiso. Para el software con el copyrigth bajo la Free Software Foundation, escriba a la Free Software Foundation; algunas veces hacemos excepciones en esto. Nuestra decisión estará guiada por los objetivos de preservar la libertad del software y de los trabajos derivados y el de promocionar el intercambio y reutilización del software en general.

SIN GARANTÍA

l) DADO QUE ESTE PROGRAMA ESTÁ LICENCIADO LIBRE DE COSTE, NO EXISTE GARANTÍA PARA EL PROGRAMA, EN TODA LA EXTENSIÓN PERMITIDA POR LA LEY APLICABLE. EXCEPTO CUANDO SE INDIQUE POR ESCRITO, LOS DUEÑOS DEL COPYRIGHT Y/O OTRAS PARTES PROVEEDORAS FACILITAN EL PROGRAMA "TAL CUAL"SI GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO, PERO NO LIMITANDO, LAS GARANTÍAS APLICABLES MERCANTILES O DE APLICABILIDAD PARA UN PROPÓSITO PARTICULAR. USTED ASUME CUALQUIER RIESGO SOBRE LA CALIDAD O LAS PRESTACIONES DEL PROGRAMA. SI EL PROGRAMA TIENE UN ERROR, USTED ASUME EL COSTE DE TODOS LOS SERVICIOS NECESARIOS PARA REPARARLO O CORREGIRLO.

m) EN NINGÚN CASO, EXCEPTO CUANDO SEA REQUERIDO POR LA LEGISLACIÓN APLICABLE O HAYA SIDO ACORDADO POR ESCRITO, NINGÚN PROPIETARIO DEL COPYRIGTH NI NINGUNA OTRA PARTE QUE MODIFIQUE Y/O REDISTRIBUYA EL PROGRAMA SEGÚN SE PERMITE EN ESTA LICENCIA SERÁ RESPONSABLE POR DAÑOS, INCLUYENDO CUALQUIER DAÑO GENERAL, ESPECIAL ACCIDENTAL O RESULTANTE PRODUCIDO POR EL USO O LA IMPOSIBILIDAD DE USO DEL PROGRAMA (CON INCLUSIÓN PERO SIN LIMITACIÓN DE LA PÉRDIDA DE DATOS O A LA GENERACIÓN INCORRECTA DE DATOS O A LAS PÉRDIDAS OCASIONADAS O POR USTED O POR TERCERAS PARTES, O A UN FALLO DEL PROGRAMA AL FUNCIONAR EN COMBINACIÓN CON CUALQUIER OTRO PROGRAMA), INCLUSO SI DICHO PROPIETARIO U OTRA PARTE HA SIDO ADVERTIDO SOBRE LA POSIBILIDAD DE DICHOS DAÑOS.

FIN DE LOS TÉRMINOS Y CONDICIONES.

10.3. Algoritmo Devorador

```
001 #include <string>
002 #include <sstream>
003 #include <set>
004 #include <iostream>
005
006 #include "Grafo.h"
007
008 using namespace std;
009
010
011 void Grafo::ConectarNodos()
012 {
013   int cont = 0, codg;
014   string nomfich =
"/home/lo/Desktop/Interfaz/Modificacion10/AsignaturaTIT.txt";
015   const char * nombre = nomfich.c_str();
016   ifstream ftit(nombre);
017
018   // Abrimos el fichero con el código y se lo asignamos al primer
nodo
019   // Le asignamos los distintos alumnos (DNI)
020   // Una vez finalizado, lo haremos tantas veces como asignaturas
haya,
021   // completaremos las conexiones comparando la interseción de
éstas.
022
023   while (ftit >> codg){
024
025     Nodos[cont].AsignarCodigo(codg);
026     stringstream n;
027     n << codg;
028
029     string nombre =
"/home/lo/Desktop/Interfaz/Modificacion10/Finales/" + n.str();
030     const char * nomfile = nombre.c_str();
031     ifstream fasig(nomfile);
032
033     while (!fasig.eof()){
034       string dni;
035       fasig >> dni;
036       Nodos[cont].alumnos.insert(dni);
037     }
038
039     cont++;
040     fasig.close();
041
042   }
043
```

```
044   ftit.close();
045
046   set <string> Inters;
047   set <string> aux;
048   set <string> result;
049   int i, j;
050
051   aux.clear();
052   result.clear();
053
054   for (i = 0; i < cont; i++){
055     for (j = 0; j < cont; j++){
056
057       if (i != j){
058
059   set_intersection(Nodos[i].alumnos.begin(),
Nodos[i].alumnos.end(), Nodos[j].alumnos.begin(),
Nodos[j].alumnos.end(), inserter(Inters, Inters.begin()));
060
061   if (Inters.empty()){
062     Conexiones[i][j] = 0;
063   }
064   else{
065
066     aux = result;
067
068     set_union(Inters.begin(), Inters.end(), aux.begin(), aux.end(),
inserter(result, result.begin()));
069
070     Conexiones[i][j] = 1;
071     Inters.clear();
072   }
073       }
074     }
075     Nodos[i].AsignarConexion(aux.size());
076   }
077 }
078
079
080 void Grafo::EscribirGrafo() const
080 {
081   Nodos.mostrar();
082 }
083
084
085 static bool TodosEnumerados(const vector<visitas> &posiciones, int
eltos)
086 {
087     int i = 0;
088
```

```
089     i = 0;
090     while (i != eltos && posiciones[i] != NO_VISITADO)
091   i++;
092
093     if (i == eltos)
094   return 1;
095     else
095   return 0;
096 }
097
098
099 void Grafo::EnumerarGrafo()
100 {
101     int eltos = Conexiones.filas();
102     vector<visitas> posiciones(eltos, NO_VISITADO);
103     int rankin = 1, max = 0; // Inicializamos el máximo al primer
elemento.
104
105     while (!TodosEnumerados(posiciones, eltos)){
106
107   while (posiciones[max] == VISITADO)
108       max++;   // Suponemos que el primer nodo no visitado es el
máximo.
109
110   for (int i = 0; i < eltos; i++){
111       if (posiciones[i] == NO_VISITADO && Nodos[max].Conexion() <
Nodos[i].Conexion())
112     max = i;
113       // Encuentra un nodo no visitado y que tiene mayor número de
conexiones que max.
114   }
115
116   posiciones[max] = VISITADO;
117   Nodos[max].AsignarPosicion(rankin); // Asignamos la posición.
118   max = 0;
119   rankin++;
120     }
121 }
122
123
124 bool Grafo::TodosColoreados()
125 {
126     int eltos = Conexiones.filas();
127     int cont = 0;
128     bool encontrado = 0;
129
130     for (int i = 0; i != eltos; i++){
131   if (Nodos[i].Color() != 0)
132       cont += 1;
133     }
```

```
134
135     /* Suponemos que no está ninguno coloreado, cuando la variable
cont
136        sea igual que el número de elementos será porque ya están
todos coloreados */
137
138     if (cont == eltos)
139   encontrado = 1;
140     else
140   encontrado = 0;
141
142     return encontrado;
143 }
144
145
146 int Grafo::ExtraeMaxVNoColor()
147 {
148     int i = 0;
149     int eltos = Conexiones.filas();
150
151     while (i != eltos && Nodos[i].Color() != 0)
152       i++;   // Suponemos como máximo el primero que no
esté coloreado.
153
154     int max = i;
155
156     for (int j = i; j < eltos; j++){
157       if (Nodos[j].Color() == 0 && Nodos[j].Posicion() <
Nodos[max].Posicion())
158   max = j;   // Buscamos el que tiene más nodos y no
esté coloreado.
159     }
160
161     return max;
162 }
163
164
165 bool Grafo::ConectadoColor(int n, int color)
166 {
167
168     int eltos = Conexiones.filas();
169     bool encontrado = false;
170
171     for (int i = 0; i != eltos; i++){
172   if (Conexiones[n][i] != 0 && Nodos[i].Color() == color)
173       encontrado = true;    //Existe un nodo conectado y con ese
color
174     }
175
176     return encontrado;
```

```
177 }
178
179
180 int Grafo::VerticeConectadoNoVisitado(int n, vector<visitas>
&marcas)
181 {
182     int eltos = Conexiones.filas();
183     int i = -1, encontrado = 0;
184
185     do{
186       i++;
187       if (i != n && marcas[i] != VISITADO && Conexiones[n][i] != 0)
188   encontrado = 1;
189       // Encontramos un vértice que está conectado pero no ha sido
visitado
190     }while (i < eltos - 1 && encontrado != 1);
191
192     if (encontrado == 0)
193       return eltos;
194     else
194       return i;
195 }
196
197
198 bool Grafo::TodosConectadosMarcados(int n, vector<visitas> &marcas)
199 {
200   int eltos = Conexiones.filas();
201   bool encontrado = true; //Suponemos que todos ya están marcados
202
203   for (int i = 0; i != eltos ; i++){
204     if (Conexiones[n][i] != 0 && marcas[i] == NO_VISITADO)
205       encontrado = false;    //Encontramos uno que no está marcado
206   }
207   return encontrado;
208 }
209
210
211 void Grafo::AsignarColorNoConectadoRecursivo(int w, int color,
vector<visitas> &marcas)
212 {
213   int eltos = Conexiones.filas();
214
215   while (!TodosConectadosMarcados(w, marcas)){
216     // Mientras que todos los nodos conectados a w no hayan sido
visitados.
217
218     int x = VerticeConectadoNoVisitado(w, marcas);
219     // Vertice que está conectado y que no ha sido visitado todavía
x.
220
```

```
221     marcas[x] = VISITADO; // Lo marcamos como visitado.
222
223     if (x < eltos){                     // Nodo válido.
224       if (!Coloreado(x)){               // No coloreado.
225   if (!ConectadoColor(x, color)){ // No conectado a un nodo
coloreado con "color".
226     AsignarColor(x, color);       // Asignamos "color".
227     AsignarColorNoConectado(x, color, marcas);
228     // Comprobamos que desde dicho nodo podemos colorear más nodos.
229   }
230   else
230     AsignarColorNoConectadoRecursivo(x, color, marcas);
231   // Si está conectado a un nodo con "color", miramos con otros
nodos conectados a él.
232       }
233       else
233   AsignarColorNoConectadoRecursivo(x, color, marcas);
234       // Si está coloreado, comprobamos con otros nodos conectados
a él.
235     }
236   }
237 }
238
239
240 void Grafo::AsignarColorNoConectado(int n, int color,
vector<visitas> &marcas)
241 {
242     vector<visitas> No_Visitados(Conexiones.filas(), NO_VISITADO);
243     int eltos = Conexiones.filas();
244     marcas[n] = VISITADO; // Lo marcamos como visitado.
245
246     while (!TodosConectadosMarcados(n, marcas)){
247       // Mientras que todos los nodos conectados a n no hayan sido
visitados
248
249       int w = VerticeConectadoNoVisitado(n, marcas);
250       // Vertice que está conectado y que no ha sido visitado
todavía w.
251
252       marcas[w] = VISITADO; // Lo marcamos como visitado.
253
254       if (w < eltos){ // Si es un nodo válido.
255   AsignarColorNoConectadoRecursivo(w, color, marcas);
256   // Comprobamos que tenga w nodos conectados que podamos colorear.
257       }
258     }
259     marcas = No_Visitados; // Inicializamos otra vez los nodos a
NO_VISITADOS.
260 }
261
```

```
262
263 bool Grafo::TodosColoreadosConectados(int v)
264 {
265   int eltos = Conexiones.filas();
266   bool encontrado = true; //Suponemos que todos ya están marcados
267
268   for (int i = 0; i != eltos ; i++){
269     if (Conexiones[v][i] != 0 && Nodos[i].Color() == 0)
270       encontrado = false;    //Encontramos uno que no está marcado
271   }
272   return encontrado;
273 }
274
275
276 int Grafo::ExtraeMaxVNoColorConexo(int v)
277 {
278     int i = 0;
279     int eltos = Conexiones.filas();
280
281     while (Conexiones[v][i] == 0 || Nodos[i].Color() != 0 && i !=
eltos)
282       i++;   // Suponemos como máximo el primero que no
esté coloreado y que esté conectado a él.
283
284     int max = i;
285
286     for (int j = i; j < eltos; j++){
287       if (Conexiones[v][i] != 0 && Nodos[j].Color() == 0 &&
Nodos[j].Posicion() < Nodos[max].Posicion())
288   max = j;   // Buscamos el que tiene más nodos, que esté conectado
a él y no esté coloreado.
289     }
290
291     return max;
292 }
293
294
295 void Grafo::ColoreoGrafosTopologico(int v)
296 {
297   int eltos = Conexiones.filas();
298   vector<visitas> marcas(eltos, NO_VISITADO); // Inicializamos los
nodos como NO_VISITADOS.
299   int color = 1;
300   int x = v;
301   AsignarColor(v, color);
302   AsignarColorNoConectado(v, color, marcas);
303
304   while (!TodosColoreadosConectados(v)){
305     if (TodosColoreadosConectados(x)){
306       color = 1;     // Cuando terminamos de colorear con un color
```

```
aumentamos y cogemos otro color.
307       x = ExtraeMaxVNoColorConexo(v);
308
309       while (ConectadoColor(x, color))
310   color ++;
311
312       AsignarColor(x, color);
313       AsignarColorNoConectado(x, color, marcas);
314       //Coloreamos con el nodo v encontrado a todos sus conexos.
315     }
316     else{
317       color++;
318       x = ExtraeMaxVNoColorConexo(x);
319       AsignarColor(x, color);
320       AsignarColorNoConectado(x, color, marcas);
321     }
322   }
323 }
324
325
326 void Grafo::ColoreoGrafos()
327 {
328   EnumerarGrafo();  // Enumeramos los nodos.
329
330   while (!TodosColoreados()){      // Mientras que no estén todos
coloreados.
331       int v = ExtraeMaxVNoColor();  // Obtenemos el que tiene mayor
grado.
332       /* Para todos los vertices conectados a v comprobamos si se
pueden colorear con "color".
333    Aumentamos el color y lo volvemos a comprobar hasta que estén
todos coloreados. Cuando todos
334    los nodos conexos a v estén coloreados, extraemos el siguiente
nodo máximo no coloreado, que
335    será un nodo no conexo a v, ya que todos los anteriores ya han
sido coloreados. */
336       ColoreoGrafosTopologico(v);
337   }
338 }
339
340
341 bool Grafo::Comprobacion()
342 {
343   int eltos = Conexiones.filas();
344   bool encontrado = false; //Suponemos que está bien coloreado.
345
346   for (int i = 0; i != eltos ; i++){
347     for (int j = 0; j != eltos; j++){
348       if (Conexiones[i][j] != 0 && (Nodos[i].Color() ==
Nodos[j].Color()))
```

```
349   encontrado = true;    //Encontramos dos nodos conectados con el
mismo color
350     }
351   }
352   return encontrado;
353 }
354
355
356 void Grafo::Modificar(int dias, int asig)
357 {
358   int min = 0, sigmin = 0, asigaux;
359
360   asigaux = asig;
361
362   while (asigaux > dias){
363
364     // Cogemos las dos asignaturas con menos coincidencias.
365
366     for (int i = 1; i != asig; i++){
367       if (Nodos[i].Conexion() < Nodos[min].Conexion()){
368   min = i;
369       }
370     }
371
372     if (min == 0)
373       sigmin = 1;
374     else
374       sigmin = 0;
375
376
377     for (int i = 0; i != asig; i++){
378       if (i != min){
379   if (Nodos[i].Conexion() < Nodos[sigmin].Conexion()){
380     sigmin = i;
381   }
382       }
383     }
384
385
386     int aux = Nodos[min].Conexion();
387     aux += Nodos[sigmin].Conexion();
388     Nodos[min].AsignarConexion(aux);
389     Nodos[sigmin].AsignarConexion(aux);
390
391     if (Nodos[min].Color() < Nodos[sigmin].Color()){
392
393       int color = Nodos[min].Color();
394
395       for(int i = 0; i != asig; i++){
396   if (Nodos[i].Color() == Nodos[sigmin].Color()){
```

```
397     Nodos[i].AsignarColor(Nodos[min].Color());
398   }
399       }
400
401       for(int i = 0; i != asig; i++){
402   if (Nodos[i].Color() > color){
403     Nodos[i].AsignarColor(Nodos[i].Color() - 1);
404   }
405       }
406     }
407     else{
408
409       int color = Nodos[sigmin].Color();
410
411       for(int i = 0; i != asig; i++){
412   if (Nodos[i].Color() == Nodos[min].Color()){
413     Nodos[i].AsignarColor(Nodos[sigmin].Color());
414   }
415       }
416
417       for(int i = 0; i != asig; i++){
418   if (Nodos[i].Color() > color){
419     Nodos[i].AsignarColor(Nodos[i].Color() - 1);
420   }
421       }
422     }
423
424     //Para cuando se salta un color, completarlo
425
426     int cont = 0, colores = 1, k = 0;
427     bool ok = false;
428
429     while (colores != dias){
430
431       while (ok == false && k < asig){
432   if (Nodos[k].Color() == colores){
433     ok = true;
434   }
435
436   k++;
437       }
438
439       if (ok != true){
440
441   for (int j = 0; j < asig; j++){
442     if (Nodos[j].Color() > colores){
443       Nodos[j].AsignarColor(Nodos[j].Color() - 1);
444     }
445   }
446
```

```
447   cont++;
448       }
449
450       colores++;
451       ok = false;
452     }
453
454     asigaux -= cont;
455     asigaux--;
456   }
457 }
458
```

10.4. Algoritmo BackTracking

```
001 #include <string>
002 #include <sstream>
003 #include <set>
004 #include <iostream>
005
006 #include "Grafo.h"
007
008 using namespace std;
009
010 void Grafo::ConectarNodos()
011 {
012   int cont = 0, codg;
013
014   string nomfich =
"/home/lo/Desktop/Interfaz/Modificacion10/AsignaturaTIT.txt";
015   const char * nombre = nomfich.c_str();
016   ifstream ftit(nombre);
017
018   // Abrimos el fichero con el código y se lo asignamos al primer
nodo
019   // Le asignamos los distintos alumnos (DNI)
020   // Una vez finalizado, lo haremos tantas veces como asignaturas
haya,
021   // completaremos las conexiones comparando la interseción de
éstas.
022
023    while (ftit >> codg){
024
025     Nodos[cont].AsignarCodigo(codg);
026     stringstream n;
027     n << codg;
028
029     string nombre =
"/home/lo/Desktop/Interfaz/Modificacion10/Finales/" + n.str();
030     const char * nomfile = nombre.c_str();
031     ifstream fasig(nomfile);
032
033     while (!fasig.eof()){
034       string dni;
035       fasig >> dni;
036       Nodos[cont].alumnos.insert(dni);
037     }
038
039     cont++;
040     fasig.close();
041
042    }
043
```

```
044    ftit.close();
045
046    set <string> Inters;
047    set <string> aux;
048    set <string> result;
049    int i, j;
050
051    aux.clear();
052    result.clear();
053
054    for (i = 0; i < cont; i++){
055      for (j = 0; j < cont; j++){
056
057        if (i != j){
058
059    set_intersection(Nodos[i].alumnos.begin(),
Nodos[i].alumnos.end(), Nodos[j].alumnos.begin(),
Nodos[j].alumnos.end(), inserter(Inters, Inters.begin()));
060
061    if (Inters.empty()){
062      Conexiones[i][j] = 0;
063    }
064    else{
065
066      aux = result;
067
068      set_union(Inters.begin(), Inters.end(), aux.begin(),
aux.end(), inserter(result, result.begin()));
069
070      Conexiones[i][j] = 1;
071      Inters.clear();
072    }
073        }
074
075        else{
076    Conexiones[i][i] = 0;
077        }
078      }
079
080      Nodos[i].AsignarConexion(aux.size());
081    }
082 }
083
084
085 void Grafo::EscribirGrafo() const
085 {
086   Nodos.mostrar();
087 }
088
089
```

```
090 bool Grafo::Conectados(int j, int k)
091 {
092   bool encontrado;
093
094   if (Conexiones[j][k] == 1)
095     encontrado = true;
096   else
096     encontrado = false;
097
098   return encontrado;
099 }
100
101
102 bool Grafo::TestColor(int k)
103 {
104   bool respuesta = true;
105   int j = 0;
106
107   while (j < k && respuesta){
108
109     if (Conectados(j, k)){
110
111       if (NodosAux[j] == NodosAux[k])
112   respuesta = false;
113     }
114     j++;
115   }
116
117   return respuesta;
118 }
119
120
121 int Grafo::ComprobarColores()
122 {
123   int i, color = 1, NCol = 0;
124   bool existe = false;
125
126   while (color != NumNodos() + 1){
127     for (i = 0; i < NumNodos(); i++){
128       if (color == NodosAux[i]){
129   existe = true;
130       }
131     }
132
133     if (existe){
134       NCol++;
135       existe = false;
136     }
137
138     color++;
```

```
139   }
140
141   return NCol;
142 }
143
144
145 void Grafo::ColoreoGrafosBacktracking(int *m, int *k)
146 {
147   int color, NumCol;
148   int n = NumNodos();
149
150   for (color = 1; color <= *m; color++){
151     AsignarColor(*k, color);
152
153     if (TestColor(*k)){
154
155       if (*k == n - 1){
156   NumCol = ComprobarColores();
157
158   if (NumCol <= *m){
159
160     for (int i = 0; i < n; i++){
161       Nodos[i].AsignarColor(NodosAux[i]);
162     }
163
164     *m = NumCol;
165   }
166       }
167       else{
168   *k = (*k + 1);
169   ColoreoGrafosBacktracking(m, k);
170       }
171     }
172   }
173 }
174
175
176 void Grafo::ColoreoGrafos()
177 {
178   int m = NumNodos();
179   int k = 0;
180
181   ColoreoGrafosBacktracking(&m, &k);
182 }
183
184
185 bool Grafo::Comprobacion()
186 {
187   int eltos = Conexiones.filas();
188   bool encontrado = false; //Suponemos que está bien coloreado.
```

```
189
190   for (int i = 0; i != eltos ; i++){
191     for (int j = 0; j != eltos; j++){
192       if (Conexiones[i][j] != 0 && (Nodos[i].Color() ==
Nodos[j].Color()))
193   encontrado = true;    //Encontramos dos nodos conectados con el
mismo color
194     }
195   }
196   return encontrado;
197 }
198
199
200 void Grafo::Modificar(int dias, int asig)
201 {
202   int min = 0, sigmin = 0, asigaux;
203
204   asigaux = asig;
205
206   while (asigaux > dias){
207
208     // Cogemos las dos asignaturas con menos coincidencias.
209
210     for (int i = 1; i != asig; i++){
211       if (Nodos[i].Conexion() < Nodos[min].Conexion()){
212   min = i;
213       }
214     }
215
216     if (min == 0)
217       sigmin = 1;
218     else
218       sigmin = 0;
219
220
221     for (int i = 0; i != asig; i++){
222       if (i != min){
223   if (Nodos[i].Conexion() < Nodos[sigmin].Conexion()){
224     sigmin = i;
225   }
226       }
227     }
228
229     int aux = Nodos[min].Conexion();
230     aux += Nodos[sigmin].Conexion();
231     Nodos[min].AsignarConexion(aux);
232     Nodos[sigmin].AsignarConexion(aux);
233
234     if (Nodos[min].Color() < Nodos[sigmin].Color()){
235
```

```
236       int color = Nodos[min].Color();
237
238       for(int i = 0; i != asig; i++){
239   if (Nodos[i].Color() == Nodos[sigmin].Color()){
240     Nodos[i].AsignarColor(Nodos[min].Color());
241   }
242       }
243
244       for(int i = 0; i != asig; i++){
245   if (Nodos[i].Color() > color){
246     Nodos[i].AsignarColor(Nodos[i].Color() - 1);
247   }
248       }
249     }
250     else{
251
252       int color = Nodos[sigmin].Color();
253
254       for(int i = 0; i != asig; i++){
255   if (Nodos[i].Color() == Nodos[min].Color()){
256     Nodos[i].AsignarColor(Nodos[sigmin].Color());
257   }
258       }
259
260       for(int i = 0; i != asig; i++){
261   if (Nodos[i].Color() > color){
262     Nodos[i].AsignarColor(Nodos[i].Color() - 1);
263   }
264       }
265     }
266
267     //Para cuando se salta un color, completarlo
268
269     int cont = 0, colores = 1, k = 0;
270     bool ok = false;
271
272     while (colores != dias){
273
274       while (ok == false && k < asig){
275   if (Nodos[k].Color() == colores){
276     ok = true;
277   }
278
279   k++;
280       }
281
282       if (ok != true){
283   for (int j = 0; j < asig; j++){
284     if (Nodos[j].Color() > colores){
285       Nodos[j].AsignarColor(Nodos[j].Color() - 1);
```

```
286     }
287   }
288
289   cont++;
290       }
291
292       colores++;
293       ok = false;
294     }
295
296     asigaux -= cont;
297     asigaux--;
298     }
299 }
300
301
302
```

10.5. Algoritmo Devorador Mejorado

```
001 #include <iostream>
002 #include <string>
003 #include <set>
004 #include <sstream>
005 #include "Grafo.h"
006
007 using namespace std;
008
009 void Grafo::ConectarNodos()
010 {
011   int cont = 0, codg;
012
013   string nomfich =
"/home/lo/Desktop/Interfaz/Modificacion10/AsignaturaTIT.txt";
014   const char * nombre = nomfich.c_str();
015   ifstream ftit(nombre);
016
017   // Abrimos el fichero con el código y se lo asignamos al primer
nodo
018   // Le asignamos los distintos alumnos (DNI)
019   // Una vez finalizado, lo haremos tantas veces como asignaturas
haya,
020   // completaremos las conexiones comparando la interseción de
éstas.
021
022   while (ftit >> codg){
023
024     Nodos[cont].AsignarCodigo(codg);
025     stringstream n;
026     n << codg;
027
028     string nombre =
"/home/lo/Desktop/Interfaz/Modificacion10/Finales/" + n.str();
029     const char * nomfile = nombre.c_str();
030     ifstream fasig(nomfile);
031
032     while (!fasig.eof()){
033       string dni;
034       fasig >> dni;
035       Nodos[cont].alumnos.insert(dni);
036     }
037
038     cont++;
039     fasig.close();
040
041   }
042
043   ftit.close();
```

```
044
045   set <string> Inters;
046   set <string> aux;
047   set <string> result;
048   int i, j;
049
050   aux.clear();
051   result.clear();
052
053   for (i = 0; i < cont; i++){
054     for (j = 0; j < cont; j++){
055
056       if (i != j){
057
058   set_intersection(Nodos[i].alumnos.begin(),
Nodos[i].alumnos.end(), Nodos[j].alumnos.begin(),
Nodos[j].alumnos.end(), inserter(Inters, Inters.begin()));
059
060   if (Inters.empty()){
061     Conexiones[i][j] = 0;
062   }
063   else{
064
065     aux = result;
066
067     set_union(Inters.begin(), Inters.end(), aux.begin(), aux.end(),
inserter(result, result.begin()));
068
069     Conexiones[i][j] = 1;
070     Inters.clear();
071   }
072       }
073     }
074     Nodos[i].AsignarConexion(aux.size());
075   }
076 }
077
078
079 void Grafo::EscribirGrafo() const
079 {
080   Nodos.mostrar();
081 }
082
083
084 int Grafo::ColoresGrafo()
085 {
086   int eltos = Conexiones.filas();
087   int maxCol = 0;
088   int color = 1;
089   bool existe = false;
```

```
090
091   for (color = 1; color < eltos; color++){
092     for (int i = 0; i < eltos; i++){
093       if (Nodos[i].Color() == color)
094   existe = true;
095     }
096
097     if (existe == true){
098       maxCol++;
099       existe = false;
100     }
101   }
102
103   return maxCol;
104 }
105
106
107 int Grafo::GradoGrafo()
108 {
109   int eltos = Conexiones.filas();
110   int maxAdy = 0;
111
112   for (int i = 0; i < eltos; i++){
113     if (Nodos[i].Color() == 1){
114       if (maxAdy < Nodos[i].Conexion())
115   maxAdy = Nodos[i].Conexion();
116     }
117   }
118
119   return maxAdy;
120 }
121
122
123 bool Grafo::ParejaVertices(int &a, int &b, int elto)
124 {
125   bool existe = false;
126   int i, j, eltos = Conexiones.filas();
127
128   for (i = 0; i < eltos; i++){
129     for (j = 0; j < eltos; j++){
130       if (i == elto)
131   i++;
132
133       if (j == elto)
134   j++;
135
136       if (i != j){
137   if ((Conexiones[i][elto] == 1 || Conexiones[j][elto] == 0) &&
(Conexiones[i][elto] == 0 || Conexiones[j][elto] == 1)){
138     a = i;
```

```
139     b = j;
140     existe = true;
141     i = Conexiones.filas();
142     j = Conexiones.filas();
143   }
144       }
145     }
146   }
147
148   return existe;
149 }
150
151
152 int Grafo::ColorDistinto(int a, int b, int c)
153 {
154   int color = 0, eltos = Conexiones.filas();
155
156   for (int elto = 0; elto < eltos; elto++){
157     if (Nodos[elto].Color() != Nodos[a].Color() &&
Nodos[elto].Color() != Nodos[b].Color() && Nodos[elto].Color() !=
Nodos[c].Color()){
158       color = Nodos[elto].Color();
159       elto = eltos;
160     }
161   }
162
163   return color;
164 }
165
166
167 bool Grafo::ConectadoColorNoNodo(int n, int color, int elto)
168 {
169
170     int eltos = Conexiones.filas();
171     bool encontrado = false;
172
173     for (int i = 0; i != eltos ; i++){
174       if (i == elto)
175   i++;
176
177       if (Conexiones[n][i] != 0 && Nodos[i].Color() == color){
178   encontrado = true;    //Existe un nodo conectado y con ese color
179       }
180     }
181     return encontrado;
182 }
183
184
185 void Grafo::Optimiza(int m, int j)
186 {
```

```
187   int eltos = Conexiones.filas();
188
189   if (m <= j){   // El grado del grafo es menor que el número de
colores empleado.
190     while (m <= j){
191
192       /* Seleccionamos los elementos con mayor grado y vamos
comprobando si se le
193    pueden asignar otro color desde m...j y así vamos eliminando
colores del
194    grafo. */
195
196       for (int elto = 0; elto < eltos; elto++){
197   if (!ConectadoColor(elto, m)){
198     AsignarColor(elto, m);
199   }
200       }
201
202       m++;
203     }
204   }
205
206   else{
207     if (m == j + 1){
208       int a, b, color, elto;
209
210       for (elto = 0; elto < eltos; elto++){
211   if(ParejaVertices(a, b, elto)){
212
213     /* Buscamos dos nodos que estén unidos sólo por un
214        vertice al nodo de mayor grado. */
215
216     color = ColorDistinto(a, b, elto);
217     // Buscamos un color distinto de los nodos.
218     if (color != 0){
219
220       if (!ConectadoColor(a, color)){
221         if (!ConectadoColorNoNodo(elto, Nodos[a].Color(), a)){
222     AsignarColor(elto, Nodos[a].Color());
223     AsignarColor(a, color);
224         }
225       }
226       else{
227         if (!ConectadoColor(b, color)){
228     if (!ConectadoColorNoNodo(elto, Nodos[b].Color(), b)){
229       AsignarColor(elto, Nodos[b].Color());
230       AsignarColor(b, color);
231     }
232         }
233       }
```

```
234     }
235   }
236       }
237     }
238   }
239 }
240
241
242 static bool TodosEnumerados(const vector<visitas> &posiciones, int
eltos)
243 {
244     int i = 0;
245
246     i = 0;
247     while (i != eltos && posiciones[i] != NO_VISITADO)
248   i++;
249
250     if (i == eltos)
251   return 1;
252     else
252   return 0;
253 }
254
255
256 void Grafo::EnumerarGrafo()
257 {
258     int eltos = Conexiones.filas();
259     vector<visitas> posiciones(eltos, NO_VISITADO);
260     int rankin = 1, max = 0; // Inicializamos el máximo al primer
elemento.
261
262     while (!TodosEnumerados(posiciones, eltos)){
263
264   while (posiciones[max] == VISITADO)
265       max++;   // Suponemos que el primer nodo no visitado es el
máximo.
266
267   for (int i = 0; i < eltos; i++){
268       if (posiciones[i] == NO_VISITADO && Nodos[max].Conexion() <
Nodos[i].Conexion())
269     max = i;
270       // Encuentra un nodo no visitado y que tiene mayor número de
conexiones que max.
271   }
272
273   posiciones[max] = VISITADO;
274   Nodos[max].AsignarPosicion(rankin); // Asignamos la posición.
275   max = 0;
276   rankin++;
277     }
```

```
278 }
279
280
281 bool Grafo::TodosColoreados()
282 {
283     int eltos = Conexiones.filas();
284     int cont = 0;
285     bool encontrado = 0;
286
287     for (int i = 0; i != eltos; i++){
288   if (Nodos[i].Color() != 0)
289       cont += 1;
290     }
291
292     /* Suponemos que no está ninguno coloreado, cuando la variable
cont
293        sea igual que el número de elementos será porque ya están
todos coloreados */
294
295     if (cont == eltos)
296   encontrado = 1;
297     else
297   encontrado = 0;
298
299     return encontrado;
300 }
301
302
303 int Grafo::ExtraeMaxVNoColor()
304 {
305     int i = 0;
306     int eltos = Conexiones.filas();
307
308     while (i != eltos && Nodos[i].Color() != 0)
309       i++;   // Suponemos como máximo el primero que no
esté coloreado.
310
311     int max = i;
312
313     for (int j = i; j < eltos; j++){
314       if (Nodos[j].Color() == 0 && Nodos[j].Posicion() <
Nodos[max].Posicion())
315   max = j;   // Buscamos el que tiene más nodos y no
esté coloreado.
316     }
317
318     return max;
319 }
320
321
```

```
322 bool Grafo::ConectadoColor(int n, int color)
323 {
324
325     int eltos = Conexiones.filas();
326     bool encontrado = false;
327
328     for (int i = 0; i != eltos; i++){
329   if (Conexiones[n][i] != 0 && Nodos[i].Color() == color)
330       encontrado = true;    //Existe un nodo conectado y con ese
color
331     }
332
333     return encontrado;
334 }
335
336
337 int Grafo::VerticeConectadoNoVisitado(int n, vector<visitas>
&marcas)
338 {
339     int eltos = Conexiones.filas();
340     int i = -1, encontrado = 0;
341
342     do{
343       i++;
344       if (i != n && marcas[i] != VISITADO && Conexiones[n][i] != 0)
345   encontrado = 1;
346       // Encontramos un vértice que está conectado pero no ha sido
visitado
347     }while (i < eltos - 1 && encontrado != 1);
348
349     if (encontrado == 0)
350       return eltos;
351     else
351       return i;
352 }
353
354
355 bool Grafo::TodosConectadosMarcados(int n, vector<visitas> &marcas)
356 {
357   int eltos = Conexiones.filas();
358   bool encontrado = true; //Suponemos que todos ya están marcados
359
360   for (int i = 0; i != eltos ; i++){
361     if (Conexiones[n][i] != 0 && marcas[i] == NO_VISITADO)
362       encontrado = false;    //Encontramos uno que no está marcado
363   }
364   return encontrado;
365 }
366
367
```

```
368 void Grafo::AsignarColorNoConectadoRecursivo(int w, int color,
vector<visitas> &marcas)
369 {
370   int eltos = Conexiones.filas();
371
372   while (!TodosConectadosMarcados(w, marcas)){
373     // Mientras que todos los nodos conectados a w no hayan sido
visitados.
374
375     int x = VerticeConectadoNoVisitado(w, marcas);
376     // Vertice que está conectado y que no ha sido visitado todavía
x.
377
378     marcas[x] = VISITADO; // Lo marcamos como visitado.
379
380     if (x < eltos){                    // Nodo válido.
381       if (!Coloreado(x)){              // No coloreado.
382   if (!ConectadoColor(x, color)){ // No conectado a un nodo
coloreado con "color".
383     AsignarColor(x, color);       // Asignamos "color".
384     AsignarColorNoConectado(x, color, marcas);
385     // Comprobamos que desde dicho nodo podemos colorear más nodos.
386   }
387   else
387     AsignarColorNoConectadoRecursivo(x, color, marcas);
388   // Si está conectado a un nodo con "color", miramos con otros
nodos conectados a él.
389       }
390       else
390   AsignarColorNoConectadoRecursivo(x, color, marcas);
391       // Si está coloreado, comprobamos con otros nodos conectados
a él.
392     }
393   }
394 }
395
396
397 void Grafo::AsignarColorNoConectado(int n, int color,
vector<visitas> &marcas)
398 {
399     vector<visitas> No_Visitados(Conexiones.filas(), NO_VISITADO);
400     int eltos = Conexiones.filas();
401     marcas[n] = VISITADO; // Lo marcamos como visitado.
402
403     while (!TodosConectadosMarcados(n, marcas)){
404       // Mientras que todos los nodos conectados a n no hayan sido
visitados
405
406       int w = VerticeConectadoNoVisitado(n, marcas);
407       // Vertice que está conectado y que no ha sido visitado
```

```
todavía w.
408
409       marcas[w] = VISITADO; // Lo marcamos como visitado.
410
411       if (w < eltos){ // Si es un nodo válido.
412   AsignarColorNoConectadoRecursivo(w, color, marcas);
413   // Comprobamos que tenga w nodos conectados que podamos colorear.
414       }
415     }
416     marcas = No_Visitados; // Inicializamos otra vez los nodos a
NO_VISITADOS.
417 }
418
419
420 bool Grafo::TodosColoreadosConectados(int v)
421 {
422   int eltos = Conexiones.filas();
423   bool encontrado = true; //Suponemos que todos ya están marcados
424
425   for (int i = 0; i != eltos ; i++){
426     if (Conexiones[v][i] != 0 && Nodos[i].Color() == 0)
427       encontrado = false;    //Encontramos uno que no está marcado
428   }
429   return encontrado;
430 }
431
432
433 int Grafo::ExtraeMaxVNoColorConexo(int v)
434 {
435     int i = 0;
436     int eltos = Conexiones.filas();
437
438     while (Conexiones[v][i] == 0 || Nodos[i].Color() != 0 && i !=
eltos)
439       i++;   // Suponemos como máximo el primero que no
esté coloreado y que esté conectado a él.
440
441     int max = i;
442
443     for (int j = i; j < eltos; j++){
444       if (Conexiones[v][i] != 0 && Nodos[j].Color() == 0 &&
Nodos[j].Posicion() < Nodos[max].Posicion())
445   max = j;   // Buscamos el que tiene más nodos, que esté conectado
a él y no esté coloreado.
446     }
447
448     return max;
449 }
450
451
```

```
452 void Grafo::ColoreoGrafosTopologico(int v)
453 {
454   int eltos = Conexiones.filas();
455   vector<visitas> marcas(eltos, NO_VISITADO); // Inicializamos los
nodos como NO_VISITADOS.
456   int color = 1;
457   int x = v;
458   AsignarColor(v, color);
459   AsignarColorNoConectado(v, color, marcas);
460
461   while (!TodosColoreadosConectados(v)){
462     if (TodosColoreadosConectados(x)){
463       color = 1;     // Cuando terminamos de colorear con un color
aumentamos y cogemos otro color.
464       x = ExtraeMaxVNoColorConexo(v);
465
466       while (ConectadoColor(x, color))
467   color ++;
468
469       AsignarColor(x, color);
470       AsignarColorNoConectado(x, color, marcas);
471       //Coloreamos con el nodo v encontrado a todos sus conexos.
472     }
473     else{
474       color++;
475       x = ExtraeMaxVNoColorConexo(x);
476       AsignarColor(x, color);
477       AsignarColorNoConectado(x, color, marcas);
478     }
479   }
480 }
481
482
483 void Grafo::ColoreoGrafos()
484 {
485   EnumerarGrafo();  // Enumeramos los nodos.
486
487   while (!TodosColoreados()){      // Mientras que no estén todos
coloreados.
488       int v = ExtraeMaxVNoColor();  // Obtenemos el que tiene mayor
grado.
489       /* Para todos los vertices conectados a v comprobamos si se
pueden colorear con "color".
490    Aumentamos el color y lo volvemos a comprobar hasta que estén
todos coloreados. Cuando todos
491    los nodos conexos a v estén coloreados, extraemos el siguiente
nodo máximo no coloreado, que
492    será un nodo no conexo a v, ya que todos los anteriores ya han
sido coloreados. */
493       ColoreoGrafosTopologico(v);
```

```
494   }
495 }
496
497
498 bool Grafo::Comprobacion()
499 {
500   int eltos = Conexiones.filas();
501   bool encontrado = false; //Suponemos que está bien coloreado.
502
503   for (int i = 0; i != eltos ; i++){
504     for (int j = 0; j != eltos; j++){
505       if (Conexiones[i][j] != 0 && (Nodos[i].Color() ==
Nodos[j].Color()))
506   encontrado = true;    //Encontramos dos nodos conectados con el
mismo color
507     }
508   }
509   return encontrado;
510 }
511
512
513 void Grafo::Modificar(int dias, int asig)
514 {
515   int min = 0, sigmin = 0, asigaux;
516
517   asigaux = asig;
518
519   while (asigaux > dias){
520
521     // Cogemos las dos asignaturas con menos coincidencias.
522
523     for (int i = 1; i != asig; i++){
524       if (Nodos[i].Conexion() < Nodos[min].Conexion()){
525   min = i;
526       }
527     }
528
529     if (min == 0)
530       sigmin = 1;
531     else
531       sigmin = 0;
532
533
534     for (int i = 0; i != asig; i++){
535       if (i != min){
536   if (Nodos[i].Conexion() < Nodos[sigmin].Conexion()){
537     sigmin = i;
538   }
539       }
540     }
```

```
541
542
543     int aux = Nodos[min].Conexion();
544     aux += Nodos[sigmin].Conexion();
545     Nodos[min].AsignarConexion(aux);
546     Nodos[sigmin].AsignarConexion(aux);
547
548     if (Nodos[min].Color() < Nodos[sigmin].Color()){
549
550       int color = Nodos[min].Color();
551
552       for(int i = 0; i != asig; i++){
553   if (Nodos[i].Color() == Nodos[sigmin].Color()){
554     Nodos[i].AsignarColor(Nodos[min].Color());
555   }
556       }
557
558       for(int i = 0; i != asig; i++){
559   if (Nodos[i].Color() > color){
560     Nodos[i].AsignarColor(Nodos[i].Color() - 1);
561   }
562       }
563     }
564     else{
565
566       int color = Nodos[sigmin].Color();
567
568       for(int i = 0; i != asig; i++){
569   if (Nodos[i].Color() == Nodos[min].Color()){
570     Nodos[i].AsignarColor(Nodos[sigmin].Color());
571   }
572       }
573
574       for(int i = 0; i != asig; i++){
575   if (Nodos[i].Color() > color){
576     Nodos[i].AsignarColor(Nodos[i].Color() - 1);
577   }
578       }
579     }
580
581     //Para cuando se salta un color, completarlo
582
583     int cont = 0, colores = 1, k = 0;
584     bool ok = false;
585
586     while (colores != dias){
587
588       while (ok == false && k < asig){
589   if (Nodos[k].Color() == colores){
590     ok = true;
```

```
591   }
592
593   k++;
594       }
595
596       if (ok != true){
597
598   for (int j = 0; j < asig; j++){
599     if (Nodos[j].Color() > colores){
600       Nodos[j].AsignarColor(Nodos[j].Color() - 1);
601     }
602   }
603
604   cont++;
605       }
606
607       colores++;
608       ok = false;
609     }
610
611     asigaux -= cont;
612     asigaux--;
613   }
614 }
615
616
617
```

Aplicación “Calendarios” programado para la elaboración de calendarios de exámenes de Titulaciones mediante la utilización de tres algoritmos empleados para el coloreo de grafos.